Robust Control for Low-Mass Quadrotors under Wind Disturbances

Denis Alpay, Nikolaj Hindsbo, Kaustabh Paul, William Kraus
{dkalpay, nhindsbo, kaustabp, wkraus}@andrew.cmu.edu

1 Abstract

This project addresses the development of robust control strategies for low-cost quadrotors operating under wind disturbances, using the Crazyflie 2.0. Three controllers—Proportional-Integral-Derivative (PID), Linear Quadratic Regulator (LQR), and Sliding Mode Control (SMC)—were designed and evaluated through a simulation-to-hardware pipeline. The Crazyflie's onboard optical flow deck enabled control without reliance on external motion capture systems, enhancing real-world applicability.

The three controllers were tested in simulated and real-world conditions, including trajectory tracking and wind disturbance scenarios. LQR exhibited exceptional orientation stability under wind, SMC demonstrated robustness despite chattering, which happens due to the fast switching between control manifolds, and the custom PID provided a reliable baseline and facilitated controller integration into the existing Crazyflie code.

This work highlights the significance of simulation-to-hardware pipelines for bridging the sim-to-real gap and demonstrates the feasibility of implementing advanced controllers on lightweight quadrotors. These findings provide valuable insights into controller trade-offs, advancing drone control for applications such as search-and-rescue and industrial inspection. The code developed and experiment videos are available in the accompanying `Github Repository`.

2 Problem Description and Previous Work

For airborne robotics projects across all domains, stability during flight is paramount to mission success. Although passive approaches to flight are possible, fast aerial maneuvers over long distances require active control and energy expenditure. Operating in atmospheric conditions introduces significant challenges, such as wind disturbances and nonlinear forces generated by propellers. Sudden environmental wind gusts or downdrafts from previous maneuvers can destabilize the quad rotors, leading to potential mission failures.

In this project, three different control strategies are developed: PID, LQR, and SMC. A simulation environment for the Crazyflie drone was designed to evaluate the controllers' stability; this also enabled low-level control of the drone and wind modeling, which allowed for a more complete testing environment for the controllers. Afterwards, these algorithms were further developed and tuned on the hardware, crossing significant hurdles with sim-to-real implementation. With the addition of a controlled fan to simulate scaled-down wind disturbances, the Crazyflie platform provided a low-cost but highly effective testbed for evaluating control robustness.

Research Gap: Existing work has primarily focused on PID and model-based controllers, since these are easier to implement and their dynamics are more common among subject matter experts [2]. Model-based controllers such as LQR and model predictive control use system models to achieve more accurate performance compared to PID alone, which is heavily influenced by errors in state measurements [5]. However, sensitivity to parameter tuning in model-based controllers and not modeling disturbances make model-based controllers insufficient in implementation alone for such conditions. Similarly, while SMC has demonstrated robustness in rejecting disturbances such as wind [4], its practical implementation on low-cost hardware remains underexplored. This gap motivates our comparative study of PID, LQR, and SMC controllers.

Our contributions are as follows: (1) Development of a simulation-to-hardware pipeline for controller design, testing, and implementation, (2) integration of robust control strategies into the Crazyflie firmware, enabling dynamic selection of PID, LQR, and SMC controllers, and (3) performance evaluation of each controller under simulated and real-world wind conditions.

Expected Outcomes: Using this simulation and hardware pipeline, the results of the paper will evaluate the performance of three different controllers. Each controller's ability to reject wind disturbances as a measure of its corresponding states will be measured on a number of different experiments, with maintaining stable flight as the quadrotor moves under wind disturbances a main factor behind the evaluation.

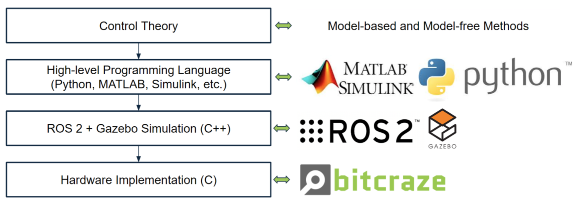

Control Theory
Model-based and Model-free Methods
High-level Programming Language (Python, MATLAB, Simulink, etc.)
MATLAB SIMULINK
python
ROS 2 + Gazebo Simulation (C++)
ROS 2
GAZEBO
Hardware Implementation (C)
bitcraze

Figure 1: Overview of the controller development pipeline used in this project. The process begins with mathematical derivations, incorporating both model-based and model-free methods. High-level programming tools such as MATLAB, Simulink, and Python were used for initial controller design and tuning. Controller prototypes were then tested in a ROS2 and Gazebo 3D simulation environment written in C++ to validate their performance under wind conditions. Finally, the controllers were implemented in hardware using the Crazyflie platform, leveraging the low-weight firmware for real-world validation. This pipeline ensures a systematic transition from theory to application, optimizing controller performance across all stages.

3 Model-Based and Model-Free Methods

3.1 System Modeling

The quadrotor was modeled as a cascaded system with decoupled dynamics, separating position control (outer loop) from attitude control (inner loop). The outer loop governs the x, y, and z positions, providing reference angles (roll, pitch, and yaw) to the inner loop, which stabilizes the quadrotor's orientation.

This cascaded approach simplifies the design by breaking complex dynamics into manageable subsystems. It also offers flexibility, allowing control blocks to be replaced or tuned independently. Additionally, it is computationally efficient, as it avoids solving coupled dynamics in real-time. The decoupling aligns naturally with the physical properties of the quadrotor, allowing precise control over the system.

The modeling approach for the Crazyflie quadrotor encompassed both nonlinear and linearized dynamics. Nonlinear models, based on rigid body assumptions, captured the full dynamics necessary for simulation accuracy. However, due to computational constraints and control complexity, linearized models were employed for controller design, focusing on hover and small-angle approximations for ideal level flight conditions. Appendix A shows the derivation of linearized dynamics in further detail.

3.2 PID

The first controller implemented was a PID controller, composed of six individual PID blocks to regulate the height, roll, pitch, and yaw components of the Crazyflie drone. The implementation of this controller served three primary purposes. First, since the controller is relatively easy to derive, testing could begin immediately to determine how exactly the controller needs to be structured for hardware implementation. The quick development time also allowed for the observation of the sim-to-real gap for the platform, which offered insight into how successful the other controllers will transition to hardware. Since the PID controller is implemented and tuned from scratch, rough estimates for control outputs and gains could be obtained and the scale in which the controller output was calculated were used for debugging the LQR and SMC algorithms.

The PID controller was implemented as a cascaded structure, shown in Figure 2. The first section of the controller contains two sub-controller algorithms, referred to as Outer Loop Control and Inner Lop Control, that take the error in X and Y position in body coordinates and output a target roll and target pitch. The Inner Loop Controller contains 4 additional PID controllers that use height, roll, pitch and yaw errors and output thrust and torque in X, Y and Z directions.

- **Outer Loop:** Position control in x and y axes, generating desired roll and pitch angles.
- **Inner Loop:** Attitude control to compute torques τ_x, τ_y, and maintain yaw at $0°$.

This cascaded control approach ensured stable hover and positional accuracy. Tuning was performed using the Crazyflie GUI client interface, which allowed for real-time adjustments without reflashing the firmware of the drone. Challenges for the iterative tuning process included balancing responsiveness and stability, particularly under wind disturbances.

3.3 LQR

The LQR is a control strategy that minimizes a cost function to achieve specific performance objectives. Unlike the PID and SMC algorithms, which use a cascaded control structure, the LQR controller is designed using the full state-space model. This approach allows the controller to simultaneously account for all system states. Derivations for the LQR controller vary widely, but most approaches for full-state feedback on a quadrotor either having control output modeled as velocities to each drone motor or torques and thrusts on the body [7] [6] [3].

The LQR controller minimizes the following cost function to balance state deviation and control effort:

$$J = \sum_{0}^{\infty} (x^\top Q x + u^\top R u) dt \tag{1}$$

where x represents the state vector, u is the control input vector.

To apply the LQR to the Crazyflie, the nonlinear dynamics of the drone were first linearized around the hover state. The resulting state-space model took the form:

$$\dot{x} = Ax + Bu$$

where A and B are the system dynamics matrices derived from the linearization process, shown in Appendix A.

The state vector x was defined as:

$$x = \begin{bmatrix} x_{\text{pos}} & y_{\text{pos}} & z_{\text{pos}} & \phi & \theta & \psi & \dot{x} & \dot{y} & \dot{z} & \dot{\phi} \\ \dot{\theta} & \dot{\psi} \end{bmatrix}^\top$$

comprising position, orientation (roll ϕ, pitch θ, and yaw ψ), along with their respective velocities. The control input vector u was defined as:

$$u = \begin{bmatrix} T & \tau_x & \tau_y & \tau_z \end{bmatrix}^\top$$

representing the torques around the x, y, and z axes as well as the total thrust for the quadrotor.

After linearization, the continuous system was converted to discrete for a control frequency of 500 Hz with a Zero-order Hold method. Initial weighting matrices Q and R were selected using Bryson's rule:

$$Q = \text{diag}\left(\frac{1}{x_{\max}^2} \frac{1}{y_{\max}^2}, \ldots\right)$$

$$R = \text{diag}\left(\frac{1}{u_{\max}^2}, \ldots\right)$$

where $x_{\max}$ and $u_{\max}$ represent the maximum allowable deviations in states and control inputs. After this calculation, these values were tuned iteratively to achieve the desired system performance.

The infinite-horizon solution to the LQR problem was computed by solving the discrete-time algebraic Riccati equation:

$$P_{t-1} = Q + A^\top P_t A - A^\top P_t B \left(B^\top P_t B + R\right)^{-1} B^\top P_t A$$

The optimal feedback gain matrix K can then be obtained using the previous calculation as well as the discretized state-space matrices:

$$K = \left(B^\top P B + R\right)^{-1} B^\top P A$$

This computation was performed offline to reduce the computational burden on the Crazyflie, allowing the drone to execute control commands efficiently during flight.

3.4 Sliding Mode Control

Sliding Mode Control is a robust control technique widely used for nonlinear systems with uncertainties and external disturbances. The core idea of SMC is to drive the system states onto a predefined surface in the state space, known as the sliding manifold, and maintain the states on this manifold thereafter. The dynamics on the manifold are designed to ensure desired system behavior, leveraging the fact that the system becomes insensitive to certain disturbances and model uncertainties when it reaches the sliding phase. A derivation of SMC was performed by the team and is shown in Appendix B. Our sliding manifold can be written as:

$$s(x_i) = \dot{x}_i + \alpha x_i$$

with x_i being an individual state and $\alpha > 0$ a tuning parameter that governs the convergence rate to the sliding manifold.

The nonlinear SMC law used in testing and simulation can be summarized by the following:

$$u_i = -\frac{1}{b}\left(f(x_i, \dot{x}_i) + \alpha \dot{x}_i\right) - k_1 s_i + k_2 \dot{x}_i^2 \,\text{sign}(s_i)$$

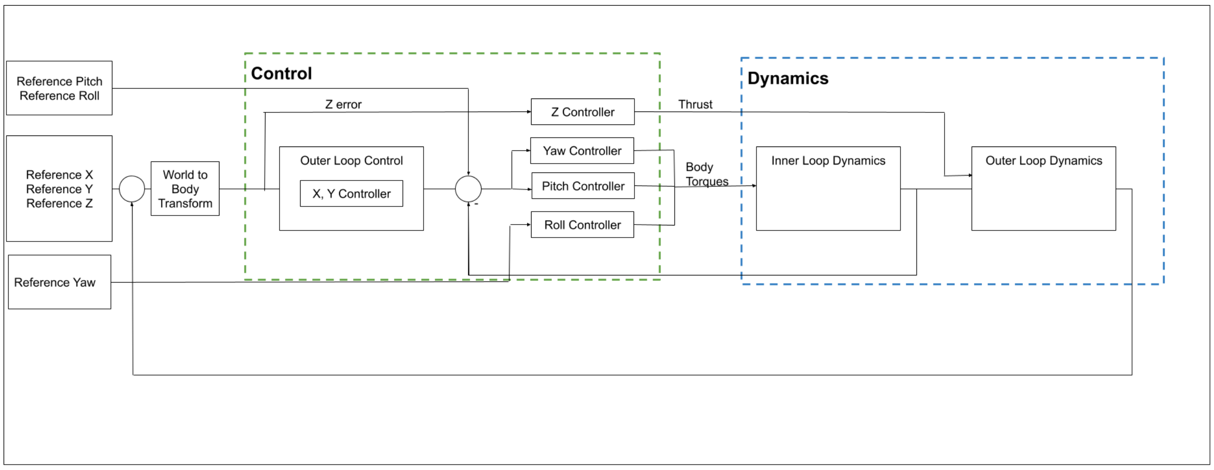


Figure 2: Controller Block Diagram for PID and SMC controllers. Reference state values are fed into the Outer and Inner Loop control blocks with additional transformations using a World to Body conversion for state error. Then, the values for computed X, Y, and Z torques on the drone body as well as the total thrust of the drone are fed into the dynamics of the system.

where:

- $-\frac{1}{b}\left(f(x_i, \dot{x}_i) + \alpha\dot{x}_i\right)$: Control term compensating for the nominal system and manifold dynamics.
- $-k_1 s_i$: Proportional term that ensures smooth convergence to the manifold.
- $-k_2 \dot{x}_i{}^2 \operatorname{sign}(s_i)$: Nonlinear switching term providing robustness and finite-time convergence.

The combined control law balances robustness and chattering reduction. The proportional term $k_1 s$ ensures smooth control near the sliding manifold, reducing chattering by avoiding discontinuous control inputs. However, relying solely on this term compromises robustness, especially under large disturbances.

The switching term $-k_2 x_i^2 sign(s)$ introduces finite-time convergence and enhances robustness by providing strong corrective action when deviations from the manifold are significant. The state-dependent scaling x_i^2 ensures that the switching term is more active during high velocities and less dominant near the manifold, minimizing unnecessary oscillations. By combining these terms, the control law provides a trade-off between robustness and chattering, ensuring reliable and smooth performance under varying conditions. We used SMC on the states z, roll ϕ and pitch ϕ.

Conclusion for Section All controllers were successfully derived, with particular attention on the cascaded structure for the SMC and PID control structures. Deriving appropriate values for the SMC was a time-consuming process, but overall effort was worthwhile to offer a more direct comparison with another model-free controller.

4 Simulation Implementation

Gazebo was chosen as the primary simulation environment for this project due to its robust physics engine, its capability to simulate wind effects, and its native support for ROS2. Alternative simulation tools were evaluated but were found inadequate due to their inability to simulate environmental disturbances, limited open-source support, or excessive complexity in setup and usage.

One key advantage of Gazebo was its flexibility in configuring physics parameters. For example, the simulation time step was set to 500 Hz to match the frequency of the hardware control loop, ensuring consistency between the simulated and real-world dynamics.

4.1 Drone Model

The Crazyflie drone was modeled in Gazebo using a Simulation Description Format (SDF) file obtained from Bitcraze's official simulation repository on GitHub [1]. The SDF file was modified to remove the default controller and provide direct speed control for each propeller. This modification allowed for the implementation of custom low-level controllers.

4.2 Control Integration

The controllers were first developed and tested in MATLAB/Simulink. This workflow is also shown in Figure 1. Once a controller was developed and validated in Matlab, it was translated into C++ for integration with Gazebo. To evaluate robustness, wind disturbances were introduced into the Gazebo environment. A uniform wind force of 1 m/s was applied to test the controllers' ability to maintain stability under external disturbances.

4.3 PID

The PID controller that was introduced in Section 3.1 was first implemented and tuned in simulation until it was able to hover and track a trajectory in X, Y, and Z directions. The tuning method was as follows.

1. Start by calculating a baseline for the height controller. This baseline allows the drone to give just enough thrust to hover when the error is 0.

2. Tune the height, roll and pitch controllers until the drone is able to hover.

3. Tune the yaw controller.

4. Add X and Y controller. The X and Y did not require much tuning, but mostly required us to find what the maximum roll and pitch targets the drone could handle.

After this, the parameter shown in Table 1 were found.

4.4 LQR

Implementation of the simulated LQR controller did not require a change in how the state space of the system was described. A stable controller with initial guesses for the infinite-horizon LQR gain matrix K was created, and the controller was fully simulated with and without wind effects. The K matrix for the simulated implementation can be found in Appendix A. This simulation was beneficial not only for reducing damage to the hardware system, but controller tuning could also be done in simulation to observe relationships between tuning different values and controller performance in wind conditions of approximately 1 meter per second.

4.5 Sliding Mode Control

Sliding Mode Control (SMC) on the quadrotor was extensively tested in simulation using various control laws and parameter configurations. This simulation-based approach allowed us to better understand the behavior of each tuning parameter and its impact on the system's performance. Through this process, we were able to design a stable SMC controller for the z-axis, roll, and pitch dynamics, which performed exceptionally well against wind disturbances and controlled the z-axis with high accuracy. Meanwhile, x, y, and yaw were controlled using PID, with x and y achieving stable control but exhibiting slower convergence compared to the SMC-controlled axes.

The inclusion of wind modeling in the simulation proved particularly useful, as it enabled us to tune parameters for different control laws and assess their robustness to external disturbances. These parameters provided a strong baseline for real-world implementation, requiring only minor retuning on the actual hardware. Overall, the simulation not only facilitated stable SMC design but also streamlined the tuning process, allowing us to achieve reliable and robust quadrotor performance in real-life scenarios.

Simulation:

- For each controller, define whether any dynamics were changed.

- Describe simulation environments

- Present results from simulations, emphasizing stability and key performance metrics.

Conclusion for Section The controller implementation in Gazebo offered a way to test the derived controllers based on previous analysis. Qualitatively, the controllers performed similarly in terms of robustness for wind. However, the ideal conditions of the simulated environment offered a proof-of-concept to determine if the control logic and rough values for tuning instead of a direct implementation; many factors contribute to this sim-to-real gap, including the lack of variation in the wind disturbance, different scaling factors for tuning, hardware motor limitations, and the lack of properly modeled aerodynamic effects such as backwashing and turbulence.

Table 1: Tuned PID Gains for Hovering and Trajectory Tracking in Simulation. This table presents the final PID gains optimized in the Gazebo simulation environment for stable hovering and accurate trajectory tracking in x, y, z, roll, pitch, and yaw. These parameters served as the initial baseline for hardware implementation, ensuring smooth transitions between simulation and real-world deployment.

Controller	$\mathbf{K_p}$	$\mathbf{K_i}$	$\mathbf{K_d}$	Integral Max	Output Max
Position X	0.2	0.0	0.01	0.0	0.2
Position Y	0.2	0.0	0.01	0.0	0.2
Height Z	0.8	0.0	0.3	0.0	0.55
Roll (ϕ)	8.0e-4	0.0	1.0e-4	0.0	1.0e-3
Pitch (θ)	8.0e-4	0.0	1.0e-4	0.0	1.0e-3
Yaw (ψ)	4e-4	0.0	2e-4	0.0	1.0e-3

5 Hardware Implementation

5.1 System Overview

The Crazyflie quadrotor served as the primary hardware platform for our project, chosen for its modular firmware and robust capabilities to implement and test custom controllers. This platform provided us with an ideal foundation for developing and evaluating the PID, SMC, and LQR controllers. The controllers were implemented as extensions to the existing firmware, leveraging Crazyflie's "Force and Torque" control mode. This allowed for direct control of thrust and torques (τ_x, τ_y, τ_z), enabling precise control over the drone's attitude and position.

The existing development of Crazyflie controllers matched how the controllers were written in the Gazebo simulation. The motivation for this approach was to minimize changes for each successive controller in the Sim-2-Real pipeline: minimizing logic errors, accelerating implementation, and integrating a controller that was working in simulation as the original guess. The integration process focused heavily on adapting controllers from a simulation environment to the Crazyflie previously derived software implementation. This involved significant adjustments to general classes, ensuring each controller compiled correctly within the firmware's make directory. Furthermore, the logic of the C++ controllers developed for simulation was converted into C for compatibility with the Crazyflie firmware. Additional modifications were made to align the simulation's position control logic with the real-world data inputs, such as those provided by the optical flow deck.

5.2 Controller Integration

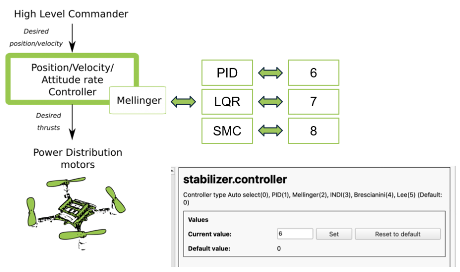
We incorporated our custom controllers into the Crazyflie firmware by modifying the control stack to include a selectable controller framework. Each controller was registered as an additional option in the "stabilizer.controller" parameter group. Specifically, the controllers were assigned unique identifiers within the parameter interface:
PID: 6, LQR: 7, SMC: 8. This enabled dynamic selection of the desired controller through the Crazyflie client without requiring firmware reflashing.

The implementation followed a cascaded control structure, similar to the Mellinger controller, but tailored to support our custom control laws. At the highest level, a commander module supplied position and velocity setpoints, which were subsequently converted into attitude rate commands by the custom controllers. These commands were finally translated into motor PWM signals through the attitude rate controller.

A visual representation of our control flow, highlighting the integration of the custom controllers, is shown in Figure 3. Controllers were added in the $src --> modules --> src --> controllers$ section of the crazyflie-firmware code stack.

5.3 Controller Tuning and Optimization

Tuning parameters for the PID and SMC controllers were simplified through the Crazyflie client's parameter interface. This allowed real-time adjustments to the controllers directly within the client interface (Figure 4). This approach significantly reduced iteration times, as no firmware flashing was required after parameter updates.


High Level Commander
Desired position/velocity
Position/Velocity/ Attitude rate Controller
Mellinger
Desired thrusts
Power Distribution motors
PID
LQR
SMC
6
7
8

stabilizer.controller

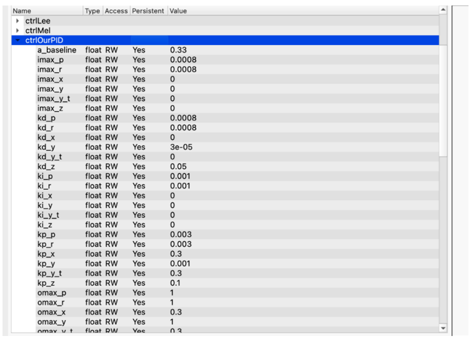
Controller type Auto select(0), PID(1), Mellinger(2), INDI(3), Brescianini(4), Lee(5) (Default: 0)

Values

Current value: 6 Set Reset to default

Default value: 0

Figure 3: Illustration of the control architecture used in the Crazyflie firmware. The flowchart demonstrates how position and velocity setpoints are processed through the commander module and subsequently fed into the custom controllers (PID, LQR, and SMC). These controllers generate attitude rate commands of thrust and torques, which are translated into motor PWM signals through the next step which is power distribution. The diagram highlights the modularity and flexibility of the system for integrating additional controllers.



Name	Type	Access	Persistent	Value
ctrlLee				
ctrlMel				
ctrlOurPID				
a_baseline	float	RW	Yes	0.33
imax_p	float	RW	Yes	0.0008
imax_r	float	RW	Yes	0.0008
imax_x	float	RW	Yes	0
imax_y	float	RW	Yes	0
imax_y_t	float	RW	Yes	0
imax_z	float	RW	Yes	0
kd_p	float	RW	Yes	0.0008
kd_r	float	RW	Yes	0.0008
kd_x	float	RW	Yes	0
kd_y	float	RW	Yes	3e-05
kd_y_t	float	RW	Yes	0
kd_z	float	RW	Yes	0.05
ki_p	float	RW	Yes	0.001
ki_r	float	RW	Yes	0.001
ki_x	float	RW	Yes	0
ki_y	float	RW	Yes	0
ki_y_t	float	RW	Yes	0
ki_z	float	RW	Yes	0
kp_p	float	RW	Yes	0.003
kp_r	float	RW	Yes	0.003
kp_x	float	RW	Yes	0.3
kp_y	float	RW	Yes	0.001
kp_y_t	float	RW	Yes	0.3
kp_z	float	RW	Yes	0.1
omax_p	float	RW	Yes	1
omax_r	float	RW	Yes	1
omax_x	float	RW	Yes	0.3
omax_y	float	RW	Yes	1
omax_y_t	float	RW	Yes	0.3

Figure 4: The Crazyflie cfclient parameter tab was utilized during the development of our custom controllers, providing a platform for tuning parameters in real-time. This image highlights the variables available for adjusting PID parameters, with similar tunable parameters implemented for the SMC controller. This feature allowed for seamless parameter adjustments even during liftoff, essential for hardware deployment.

In contrast, the LQR controller required offline computation of the optimal feedback gain matrix (K) by solving the Riccati equation in Python. This matrix was then hard coded into the firmware, necessitating a firmware flash for each update. While less flexible, this approach ensured computational efficiency during real-time operation.

Controller performance was primarily evaluated through iterative testing. For the initial guess of the values, the controllers that were stable in hovering and takeoff in simulation were used. From there, observational tuning involved modifying controller gains based on the drone's ability to stabilize in real life. For example, roll-based terms were adjusted when the drone exhibited instability in maintaining level flight, while altitude gains were refined to minimize overshoot and achieve consistent height control.

5.4 Hardware Final Controllers Summary

The final controller tuning parameters are shown on Table 2. In hardware deployment controller output had separate minimum and maximum values whereas in simulation minimum was just negative of maximum. The reason for that change is that we implemented a positive minimum for the Z controller in order to have a controller descent when the drone is too high.

LQR Controller

The gain matrix (K) for the final LQR controller is:

K = $10^{-3}\times$

$$\begin{bmatrix} 0 & 0 & 70 & 0 & 0 & 0 & 0 & 0 & 406.9 & 0 \\ 0 & 0 & & & & & & & & \\ 0 & 0 & 0 & 5.62 & 0 & 0 & 0 & 0 & 0 & 0.95 \\ 0 & 0 & & & & & & & & \\ 0.86 & 0 & 0 & 0 & 5.65 & 0 & 1.32 & 0 & 0 & 0 \\ 0.95 & 0 & & & & & & & & \\ 0 & -0.86 & 0 & 0 & 0 & 8.98 & 0 & -1.32 & 0 & 0 \\ 0 & 1.09 & & & & & & & & \end{bmatrix}$$

Table 3: Final controller parameters for SMC implemented on hardware. This table summarizes the cascaded control structure used in the final hardware implementation, where SMC was applied to the z (height), roll, and pitch dynamics to leverage its robustness against disturbances, while PID controllers were retained for x, y, and yaw control to ensure simplicity and stability in these axes. The parameters reflect the final tuning achieved through iterative testing under real-world conditions, balancing robustness, precision, and minimizing the effects of chattering.

Controller	Type	Parameters and Range
X-Controller	PID	$k_p = 1.0, k_i = 0.0, k_d = 0.0$, Range: $[-0.2, 0.2]$
Y-Controller	PID	$k_p = 1.0, k_i = 0.0, k_d = 0.0$, Range: $[-0.2, 0.2]$
Z-Controller	SMC	$\alpha_z = 25.0, \text{gain} = 1.0, m = 0.031, A = 0.35$
Yaw-Controller	PID	$k_p = 0.001, k_i = 0.0, k_d = 3 \times 10^{-5}$, Range: $[-1.0, 1.0]$
Roll-Controller	SMC	$k_1 = 0.003, k_2 = 1e-5, \text{gain} = 0.13, \alpha_{\text{roll}} = 0.8$
Pitch-Controller	SMC	$k_1 = 0.003, k_2 = 1e-5, \text{gain} = 0.19, \alpha_{\text{pitch}} = 0.8$

Conclusion for Section The implementation of the hardware was greatly aided by the availabil-

Table 2: Final Tuned PID Gains and Output Ranges for Hardware Deployment, Optimized for Hover Stability and Trajectory Tracking in Real-World Conditions. The gains ($\mathbf{K_p}$, $\mathbf{K_i}$, $\mathbf{K_d}$) and output ranges were iteratively tuned to ensure a balance between responsiveness, stability, and robustness, particularly under wind disturbances.

Controller	$\mathbf{K_p}$	$\mathbf{K_i}$	$\mathbf{K_d}$	**Range**
X-Controller	0.3	0.0	0.0	$[-0.3, 0.3]$
Y-Controller	0.3	0.0	0.0	$[-0.3, 0.3]$
Z-Controller	0.1	0.0	0.05	$[0.25, 0.4]$
Yaw-Controller	0.001	0.0	3×10^{-5}	$[-1.0, 1.0]$
Roll-Controller	0.003	0.001	0.0008	$[-1.0, 1.0]$
Pitch-Controller	0.003	0.001	0.0008	$[-1.0, 1.0]$

ity of a custom control stack modified from existing implementations. Preliminary tuning values from the initial Gazebo simulation were helpful for the PID controller especially, but LQR tuning was more effective via hand-tuning on hardware. All controllers were successfully integrated, with subsequent controller development and tuning time reduced with each subsequent implementation.

6 Testing Environment

6.1 Experimental Setup

Testing was conducted in a controlled indoor environment (the Carnegie Mellon Drone Cage) to evaluate the performance of the custom controllers under two primary scenarios:

1. **Square Wave Trajectory Tracking:** The drone was commanded to follow a predefined trajectory consisting of diagonal, lateral, and vertical movements of 0.1 - 0.25 meters.

2. **Hover Stability in Wind Conditions:** A fan was used to simulate wind disturbances at a fixed distance from the drone's take-off (the take-off point was the same for all controllers) and measured during the trial to ensure it consistently remained around 1 meter per second.

6.2 Evaluation Metrics

Performance was evaluated using the following metrics:

- **Position Error:** The Euclidean distance between the desired and actual positions along the x, y, and z axes, providing a measure of positional accuracy.
- **Attitude Error:** The combined angular deviations in roll, pitch, and yaw. This metric was primarily used to assess system stability during hover tests under wind disturbances.

Normalized error, used for both position and attitude metrics, was computed as follows:

$$\text{Normalized Error} = \frac{\sqrt{\sum_{i=1}^{n}(\text{error}_i^2)}}{\sqrt{\sum_{i=1}^{n}(\text{target}_i^2) + 10^{-6}}} \tag{2}$$

The small constant 10^{-6} ensures numerical stability in cases where all target$_i = 0$, such as when tracking orientation errors $(\text{roll}, \text{pitch}, \text{yaw} = 0)$.

6.3 Subjective Observations

During tuning, the following observations were noted:

PID Controller: Initial tuning of the PID controller presented challenges, but once a stable configuration was achieved, fine-tuning became more intuitive. The PID controller exhibited excellent performance for position tracking and hovering accuracy. However, it was the least robust under wind disturbances, often exhibiting instability during gusts.

LQR Controller: The LQR controller was the most challenging to tune, as stability was highly sensitive to the gain matrix (K). For a long time, the drone failed to stabilize or take off, even with a gain matrix that was working for simulation. Leveraging insights from prior work provided by Dr. Bedillion, we used a reference K matrix to approximate optimal magnitudes for tuning the Q and R matrices. After iterative adjustments, including visual inspection and manual refinement of thrust terms, the LQR controller achieved exceptional stability in wind conditions, maintaining flight even under strong disturbances; although, this controller was not able to properly regulate the position of the drone and often drifted away from the target X and Y states. Despite its robustness, the controller showed susceptibility to drift during position tracking, prioritizing roll and pitch stability over precise positional control. The tuning process for the LQR controller was more time-intensive compared to the other controllers, as each adjustment to the K matrix required reflashing the firmware and re-solving the Riccati equation. Consequently, tuning often involved larger incremental changes, which favored improving orientation stability—critical for wind testing—at the expense of precise position control. Despite these challenges, the LQR controller demonstrated robust performance under windy conditions, highlighting its potential for tasks where maintaining orientation is paramount.

SMC Controller: From the outset, the SMC controller demonstrated promising stability during initial liftoff tests, guided by simulation-based parameters. However, achieving a balance between positional accuracy and robustness proved challenging. The SMC controller excelled in handling extreme disturbances, with rapid counter-movements effectively maintaining flight. Nevertheless, its tendency to prioritize stability over precise position control resulted in suboptimal hovering performance. Fine-tuning involved iterative trade-offs among chatter reduction, position control, and disturbance rejection, making the process complex and time-intensive.

7 Results

7.1 Square Wave Tracking

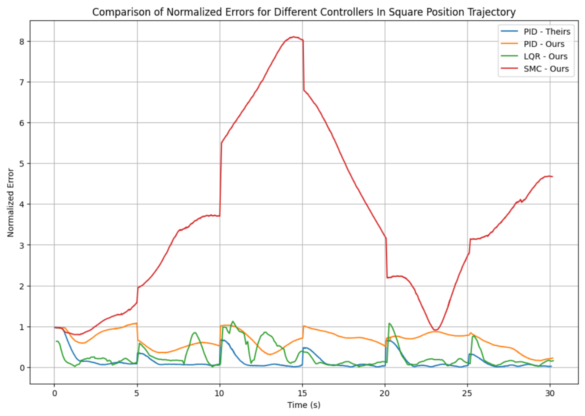


Figure 5: Comparison of Position Normalized Errors for Different Controllers in Square Wave Position Trajectory Tracking. The figure highlights the performance of the controllers over time, showcasing the ability to handle sharp transitions and maintain tracking accuracy over time.

The square wave tracking test evaluates how well each controller can maintain position accuracy in a trajectory with sharp transitions – with limited time for mistakes. From Figure 5, the following observations can be made:

- **PID (Theirs):** This controller shows stable performance with moderate error levels throughout the trajectory. Their controller was able to make sharp transitions and avoid overshooting errors, with acceptable hovering performance.

- **PID (Ours):** Our PID implementation exhibits higher errors. Qualitatively, the sharp left, right, and especially the corner tracking showed the x-y controllers could benefit from tuning. However, given the scope of the project this controller showed acceptable performance and if given more time to tune could in theory get as good or better as Crazyflie's PID controller.

- **LQR (Ours):** LQR demonstrates the best overall performance of the custom controllers developed with consistently low normalized errors. LQR was faster to react to position changes, showing sharp changes in trajectory – but sometimes overshot. If position error was minimized, its hover mode was qualitatively and quantitatively shown to be smooth.

- **SMC (Ours):** The SMC controller struggles significantly with position control, leading to large spikes in error. This suggests that SMC may require further tuning or a more robust design to handle high-frequency trajectory changes effectively. Perhaps, if it had used its own SMC controller for x and y position control improvements could have been seen.

7.2 Hover With Wind Conditions

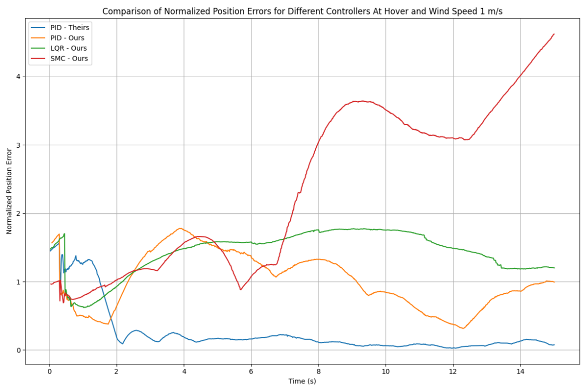


Figure 6: Comparison of Normalized Position Errors for Different Controllers During Hover Test Under Wind Conditions (1 m/s). The figure highlights how each controller maintains position stability despite disturbances.

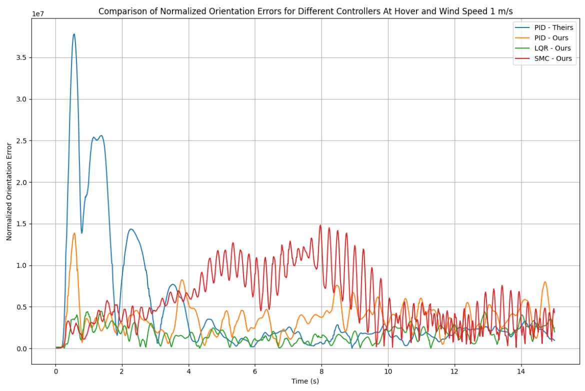


Figure 7: Comparison of Normalized Orientation Errors for Different Controllers During Hover Test Under Wind Conditions (1 m/s). The figure evaluates how well the controllers manage angular stability in the presence of wind disturbances.

The hover test under wind disturbances evaluates the controllers' abilities to maintain position and orientation stability. Key insights from Figures 6 and 7 are as follows:

Position Error Analysis

- **PID (Theirs):** In takeoff, the PID struggled at first, but regained its balance – maintaining a consistently small position error.
- **PID (Ours):** Our PID controller has worse performance than the Crazyflie implementation, but showed effort to control position.
- **LQR (Ours):** LQR showed stability in roll and pitch, but came with a constant price in position. LQR showed effort to maintain position, but prioritized orientation stability.
- **SMC (Ours):** SMC performs better in hover tests than in the square wave trajectory but still exhibits higher position errors compared to LQR and PID, particularly as the test continued.

Orientation Error Analysis

- **PID (Theirs):** At liftoff, their PID displays the largest orientation errors, prioritizing position control over potential tipping. However, it maintained hover and recovered well after time – outperforming all controllers except the LQR.
- **PID (Ours):** Our PID implementation reduces orientation errors compared to the baseline version, providing improved angular stability.
- **LQR (Ours):** LQR excelled here, maintaining the lowest orientation errors – even smaller than PID. This is a promising result that shows our controller could be more robust to wind disturbances than the in-built software PID.
- **SMC (Ours):** SMC shows moderate performance, with higher variability in orientation errors compared to other controllers - likely due to chattering which was evident. With more time, tuning could have been done to minimize chattering in wind conditions.

Summary of Square Wave Tracking and Hover with Wind Conditions

The evaluation of controllers in square wave tracking and hover under wind disturbances revealed key performance differences:

itemize

PID (Theirs): Consistently reliable in both tests, maintaining moderate errors and robust recovery during hover.

PID (Ours): Showed acceptable performance with room for improvement in tuning to match or exceed the baseline.

LQR (Ours): Demonstrated the best orientation stability and low tracking errors but occasionally overshot and traded position accuracy for orientation stability

SMC (Ours): Struggled in trajectory tracking and exhibited chattering under wind, requiring significant tuning to improve stability.

Qualitative Further Observations

Additional testing with increasing wind speeds in the drone cage provided insights into the controllers' robustness under extreme disturbances. The custom PID controller maintained stability only at low wind speeds, while the SMC handled low to medium speeds but failed at higher intensities. The baseline PID resisted high wind speeds but ultimately lost control.

The LQR controller exhibited the best performance, maintaining stability even at the highest wind setting. However, position drift led the drone to collide with the wall, underscoring a trade-off between orientation stability and positional control. These observations highlight the controllers' varying capacities to handle real-world disturbances and the potential for further refinement.

Overall, the LQR excelled in orientation stability, while their PID remained a robust baseline for position control. Our SMC and PID demonstrated potential but require further optimization. Given more time, we believe an implementation of SMC and LQR would be ideal for improved controllers in wind conditions. Additionally, LQR could be further tuned to find a more optimal tradeoff between orientation control and position control.

8 Reflection and Next Steps

The team encountered several hardware challenges during implementation on the Crazyflie platform. Broken propellers, damaged chassis, unresponsive optical flow decks, and faulty motor interface plugs frequently interrupted testing. Furthermore, sensor drift in the optical flow deck and the pitch angle being defined in a left-handed manner introduced confusion during tuning, resulting in numerous false logic errors. Tuning controllers on hardware was particularly time-intensive, as experiments required reflashing firmware for parameter updates, with test durations ranging from seconds to over ten minutes.

Despite these challenges, the systematic simulation-to-hardware pipeline enabled the successful deployment of all three controllers. This pipeline provided insights into the sim-to-real gap, emphasizing the effects of unmodeled dynamics and physical discrepancies. Notably, the SMC controller demonstrated high potential for rejecting disturbances, making it a promising candidate for further refinement. Future efforts will focus on fine-tuning SMC to address chattering, improve positional accuracy, and explore hybrid control strategies combining the robustness of SMC with the precision of LQR. Additionally, incorporating advanced adaptive control methods or enhanced modeling techniques may better account for real-world dynamics, streamlining the transition from simulation to hardware.

9 Conclusion

This study demonstrated the successful implementation of SMC on a low-weight quadrotor platform like the Crazyflie, which, to the best of our knowledge, had not been previously achieved. In addition, three distinct controllers— PID, LQR, and SMC — were designed, implemented, and evaluated, showcasing characteristics consistent with theoretical expectations, such as chattering in SMC and robust disturbance rejection in LQR. These results validated the practical application of advanced control strategies on lightweight drones using only onboard sensors, such as Crazyflie's optical flow deck, without relying on external systems like motion capture rooms or lighthouse tracking.

Beyond the technical contributions, this project provided a valuable learning experience for us as master's students entering the field of control systems and aerial robotics. By navigating challenges such as hardware limitations, unmodeled dynamics, and the sim-to-real gap, we gained critical insights into the complexities of designing robust controllers for real-world applications. The study also emphasized the importance of systematic simulation-to-hardware pipelines, which facilitated iterative development and reduced the risk of integration errors.

The ability to develop and test controllers using localized onboard sensors highlights the potential for deploying drones in real-world conditions, where reliance on controlled environments is not feasible. This work serves as a stepping stone for expanding autonomous aerial robot applications into scenarios such as search-and-rescue missions, industrial inspections, and other tasks requiring robust and adaptive control in dynamic and unpredictable environments.

Reproducibility: All videos and source code, including Crazyflie and simulation scripts, are available on our `Github Repository`.

Acknowledgments

We thank Dr. Bedillion, Karthik Karumanchi, and Arvind Car for guidance and support.

Appendix A: System Variables and Nonlinear to Linearized Dynamics

Symbol	Description
x, y, z	Position of the drone in 3D space
ϕ, θ, ψ	Roll, pitch, and yaw angles
I_x, I_y, I_z	Inertia around respective axes
U_1, U_2, U_3, U_4	Thrust and torques generated by the propellers
A_x, A_y, A_z	Drag coefficients for the drone

Nonlinear Dynamics

The nonlinear dynamics of the drone are:

$$
\begin{aligned}
\ddot{x} &= \frac{U_1}{m}(\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi) - A_x\dot{x} \\
\ddot{y} &= \frac{U_1}{m}(\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi) - A_y\dot{y} \\
\ddot{z} &= \frac{U_1}{m}\cos\phi\cos\theta - g - A_z\dot{z} \\
\ddot{\phi} &= \dot{\theta}\dot{\psi}(\frac{I_y - I_z}{I_x}) - \frac{J_R}{I_x}\dot{\theta}\Omega_R + \frac{1}{I_x}U_2 \\
\ddot{\theta} &= \dot{\phi}\dot{\psi}(\frac{I_z - I_x}{I_y}) - \frac{J_R}{I_y}\dot{\phi}\Omega_R + \frac{1}{I_y}U_3 \\
\ddot{\psi} &= \dot{\phi}\dot{\theta}(\frac{I_x - I_y}{I_z}) + \frac{1}{I_z}U_4
\end{aligned}
$$

Linearized Dynamics

Linearizing the dynamics using small-angle approximations and neglecting higher-order terms:

$$
\begin{gathered}
\ddot{x} \approx g\theta, \quad \ddot{y} \approx -g\phi, \quad \ddot{z} \approx \frac{\Delta U_1}{m}, \\
\ddot{\phi} \approx \frac{1}{I_x}U_2, \quad \ddot{\theta} \approx \frac{1}{I_y}U_3, \quad \ddot{\psi} \approx \frac{1}{I_z}U_4.
\end{gathered}
$$

From these equations, one can find the A and B matrices using their respective Jacobians.

LQR Gain Matrix for Gazebo Simulation

The optimal infinite-horizon LQR gain matrix K for the LQR controller in Gazebo simulation is:

$$
K = \begin{bmatrix}
-1.9e-13 & 0 & 12 & 0 & -2.9e-12 & 0 & -3.6e-13 & 0 & 0.9 & 0 \\
-2.3e-12 & 0 \\
0 & -0.00159383354 & 0 & 0.0148803678 & 0 & 1.36528461 \times 10^{-11} & 0.00000000 & -0.00219927784 & 0.00000000 & 0.00514554485 \\
0.00000000 & 7.18928267 \times 10^{-14} \\
0.00159383355 & 0.00000000 & -2.28114205 \times 10^{-13} & 0.00000000 & 0.0148803679 & 0.00000000 & 0.00219927786 & 0.00000000 & -8.70169129 \times 10^{-15} & 0.00000000 \\
0.00514554485 & 0.00000000 \\
0.00000000 & -1.31704745 \times 10^{-12} & 0.00000000 & 1.60295462 \times 10^{-11} & 0.00000000 & 0.0406755758 & 0.00000000 & -4984516 \times 10^{-12} & 0.00000000 & 5.91393329 \times 10^{-14} \\
0.00000000 & 0.00566081644
\end{bmatrix}
$$

Appendix B: Sliding Mode Controller derivation

.1 Introduction

SMC is a robust control method designed for nonlinear systems with uncertainties or external disturbances. It works by driving the system states to a predefined surface in the state space, called the **sliding manifold**, and maintaining the states on this manifold. Once the system reaches the sliding manifold, the dynamics are constrained to follow it, resulting in desired behavior and robustness to system uncertainties.

Key characteristics of SMC:

- **Robustness:** SMC is highly robust to uncertainties and disturbances, making it suitable for systems with unmodeled dynamics.
- **Sliding Phase:** Once on the sliding manifold, the system behaves according to the desired dynamics.
- **Finite-Time Convergence:** The system states are driven to the manifold in finite time.

Despite these advantages, traditional SMC can lead to high-frequency control signal oscillations, known as **chattering**, which must be mitigated for practical implementation.

.2 Sliding manifold

The sliding manifold is a carefully chosen surface in the system's state space that defines the desired dynamic behavior. For a single-input, single-output system, the sliding manifold is typically defined as:

$$s(x) = \dot{x} + \alpha x$$

where:

- x is the system state,
- $\alpha > 0$ is a design parameter that controls the rate of convergence.

The goal of SMC is to design a control law that ensures:

1. The system state reaches the sliding manifold in finite time ($s(x) = 0$).
2. The system remains on the manifold for all subsequent times.

This is achieved by designing the control law to satisfy a Lyapunov stability condition.

.3 Lyapunov stability

In control theory, Lyapunov stability is used to verify the stability of a system by defining a scalar energy-like function $V(s)$, known as a **Lyapunov function**. The Lyapunov function measures how "far" the system is from the desired equilibrium. By ensuring that $V(s)$ decreases over time ($\dot{V}(s) \leq 0$), we can guarantee that the system will converge to and remain on the sliding manifold.

For SMC, the Lyapunov function is chosen as:

$$V(s) = \frac{1}{2}s^2$$

This function satisfies the following properties:

- **Positive Definiteness:**
$$V(s) > 0 \quad \text{for } s \neq 0, \quad V(s) = 0 \text{at } s = 0$$
- **Radial Unboundedness:**
$$V(s) \to \infty \quad \text{as } |s| \to \infty$$
- **Negative Semi-Definiteness of the Derivative:**
$$\dot{V}(s) = s\dot{s} \leq 0$$

These properties ensure that the system states are driven toward the sliding manifold and remain there, guaranteeing global asymptotic stability.

.4 General control law derivation

The time derivative of the Lyapunov function is:

$$\dot{V}(s) = s\dot{s}$$

Using the definition of the sliding manifold:

$$\dot{s}(x) = \ddot{x} + \alpha\dot{x}$$

Substitute the system dynamics:

$$\ddot{x} = f(x, \dot{x}) + bu$$

where $f(x, \dot{x})$ represents the nonlinear dynamics and b is the control gain. This gives:

$$\dot{s}(x) = f(x, \dot{x}) + bu + \alpha\dot{x}$$

Substitute $\dot{s}$ into $\dot{V}(s)$:

$$\dot{V}(s) = s\left(f(x, \dot{x}) + bu + \alpha\dot{x}\right)$$

To ensure stability ($\dot{V}(s) \leq 0$), the control input u is designed as:

$$u = u_{\text{dyn}} + u_{\text{sw}}$$

The first control law u_{dyn} is responsible for compensating the system dynamics and can be written as:

$$u_{\text{eq}} = -\frac{1}{b}\left(f(x, \dot{x}) + \alpha\dot{x}\right)$$

.

It is responsible for compensating the dynamics component, since the dynamics cannot guarantee negative semi-definiteness. For the second control law we can try following approaches.

1. Switching Control (u_{sw})

$$u_{\text{sw}} = -k\,\text{sign}(s)$$

The combined control law becomes:

$$u = -\frac{1}{b}\left(f(x, \dot{x}) + \alpha\dot{x}\right) - k\,\text{sign}(s)$$

Substitute u into $\dot{s}(x)$:

$$\dot{s}(x) = -kb\,\text{sign}(s)$$

Substitute into $\dot{V}(s)$:

$$\dot{V}(s) = -kb|s|$$

Since $\dot{V}(s) < 0$ for all $s \neq 0$, the system is globally asymptotically stable.

The downside to this approach is that the discontinuity in the sign function and constant gain cause high frequency stationary chattering. In general, we know that control laws that solely depend on such switching terms are robust but suffer from severe chattering due to the discontinuous nature of the control input.

2. State-dependent gain-based switching control

$$u_{\text{sw}} = -k_1 \dot{x}^2 \operatorname{sign}(s)$$

The combined control law becomes:

$$u = -\frac{1}{b}\left(f(x,\dot{x}) + \alpha\dot{x}\right) + k_1 \dot{x}^2 \operatorname{sign}(s)$$

Substitute u into $\dot{s}(x)$:

$$\dot{s}(x) = bk_1\dot{x}^2 \operatorname{sign}(s)$$

Substitute into $\dot{V}(s)$:

$$\dot{V}(s) = bk_1\dot{x}^2 s \operatorname{sign}(s)$$

In this case we can see that the state-dependent scaling term reduces the severity of chattering but can amplify nonlinearities when the system velocity $\dot{x}$ is high. It also exhibits weaker control at low velocities, delaying convergence to the sliding manifold. Even though the state dependent gain reduces intensity of chattering it is still prevalent.

3. Manifold-dependent smooth control Law

$$u_{\text{sw}} = -k_1 s$$

The combined control law becomes:

$$u = -\frac{1}{b}\left(f(x,\dot{x}) + \alpha\dot{x}\right) - k_1 s$$

Substitute u into $\dot{s}(x)$:

$$\dot{s}(x) = -k_1 b s$$

Substitute into $\dot{V}(s)$:

$$\dot{V}(s) = -k_1 b s^2$$

Since $\dot{V}(s) < 0$ for all $s \neq 0$, the system is globally asymptotically stable. This control law eliminates chattering entirely but converges slower than previous methods, making it less robust as well. The reason is that this controller only reaches the manifold asymptotically, while switching controllers manage to do it in finite time.

4. Combined Control Law

$$u_{\text{sw}} = -k_1 s - k_2 \dot{x}^2 \operatorname{sign}(s)$$

The combined control law becomes:

$$u = -\frac{1}{b}\left(f(x,\dot{x}) + \alpha\dot{x}\right) - k_1 s - k_2 \dot{x}^2 \operatorname{sign}(s)$$

Substitute u into $\dot{s}(x)$:

$$\dot{s}(x) = -k_1 b s - bk_2\dot{x}^2 \operatorname{sign}(s)$$

Substitute into $\dot{V}(s)$:

$$\dot{V}(s) = -k_1 b s^2 - bk_2\dot{x}^2 s \operatorname{sign}(s)$$

Again we can see that $\dot{V}(s) < 0$ for all $s \neq 0$, which means that the system is globally asymptotically stable.

The control parameters k_1 and k_2 play a crucial role in determining the behavior of the system under the SMC framework The parameter k_1, associated with the proportional term, directly affects the rate of convergence to the sliding manifold. A larger k_1 leads to faster convergence and greater damping but can also result in overshoot or oscillations if it is too high. On the other hand, a smaller k_1 results in slower convergence, which smoothens the system's response but may reduce its ability to reject disturbances effectively. Similarly, k_2, associated with the switching term, governs the robustness of the system to uncertainties and external disturbances. A larger k_2 enhances robustness but increases the likelihood of chattering, a high-frequency oscillation that can damage actuators or reduce efficiency. A smaller k_2, while reducing chattering and ensuring smoother control, may weaken the system's disturbance rejection capabilities.

Appendix C: Simulation Setup and Tools

Gazebo Simulation

The Gazebo environment was configured using the CrazyFlie URDF model and open-source plugins. Wind disturbances were simulated with the `crazyflie_world.sdf` plugin, allowing dynamic parameter adjustments.

ROS2 Workflow

The ROS2 framework was used to control and monitor the simulation:

- `ros2 node list, ros2 topic list`: Debugging tools.
- Custom scripts for motor control and sensor feedback.

Simulation Parameters

Key parameters used in the simulation include:

- Wind speed: 0-3 m/s

References

[1] bitcraze. *GitHub - bitcraze/crazyflie-simulation*. GitHub, 2022. URL: https://github.com/bitcraze/crazyflie-simulation (visited on 12/14/2024).

[2] Bo Dai et al. “Wind Disturbance Rejection for Unmanned Aerial Vehicle Based on Acceleration Feedback Method”. In: *2018 IEEE Conference on Decision and Control (CDC)*. 2018, pp. 4680–4686. DOI: 10.1109/CDC.2018.8619798.

[3] Zachary T. Dydek, Anuradha M. Annaswamy, and Eugene Lavretsky. “Adaptive Control of Quadrotor UAVs: A Design Trade Study With Flight Evaluations”. In: *IEEE Transactions on Control Systems Technology* 21.4 (2013), pp. 1400–1406. DOI: 10.1109/TCST.2012.2200104.

[4] Arie Levant. “Sliding mode control and observation”. In: *Boris J. Lurie Memorial Conference on Sliding Mode Control*. Springer. 2017, pp. 22–35. URL: https://inria.hal.science/hal-01566857.

[5] Khai Nguyen et al. *TinyMPC: Model-Predictive Control on Resource-Constrained Microcontrollers*. 2024. arXiv: 2310.16985 [cs.RO]. URL: https://arxiv.org/abs/2310.16985.

[6] Robert Scholten. *Modelling and control of quadcopter*. Tech. rep. University of Twente, 2017. URL: https://d1wqtxts1xzle7.cloudfront.net/53671248/eluu11_public-libre.pdf.

[7] Zaid Tahir, Waleed Tahir, and Saad Ali Liaqat. “State Space System Modeling of a Quad Copter UAV”. In: *CoRR* abs/1908.07401 (2019). arXiv: 1908.07401. URL: http://arxiv.org/abs/1908.07401.