INTER IIT
TECH MEET 14.0

DRONA™ AVIATION

Team 24

Inter IIT Tech Meet 14.0

CONTENTS

PROBLEM UNDERSTANDING

The aim is to improve the stability of indoor nano-drones **without using GPS**.

MAIN TASKS:

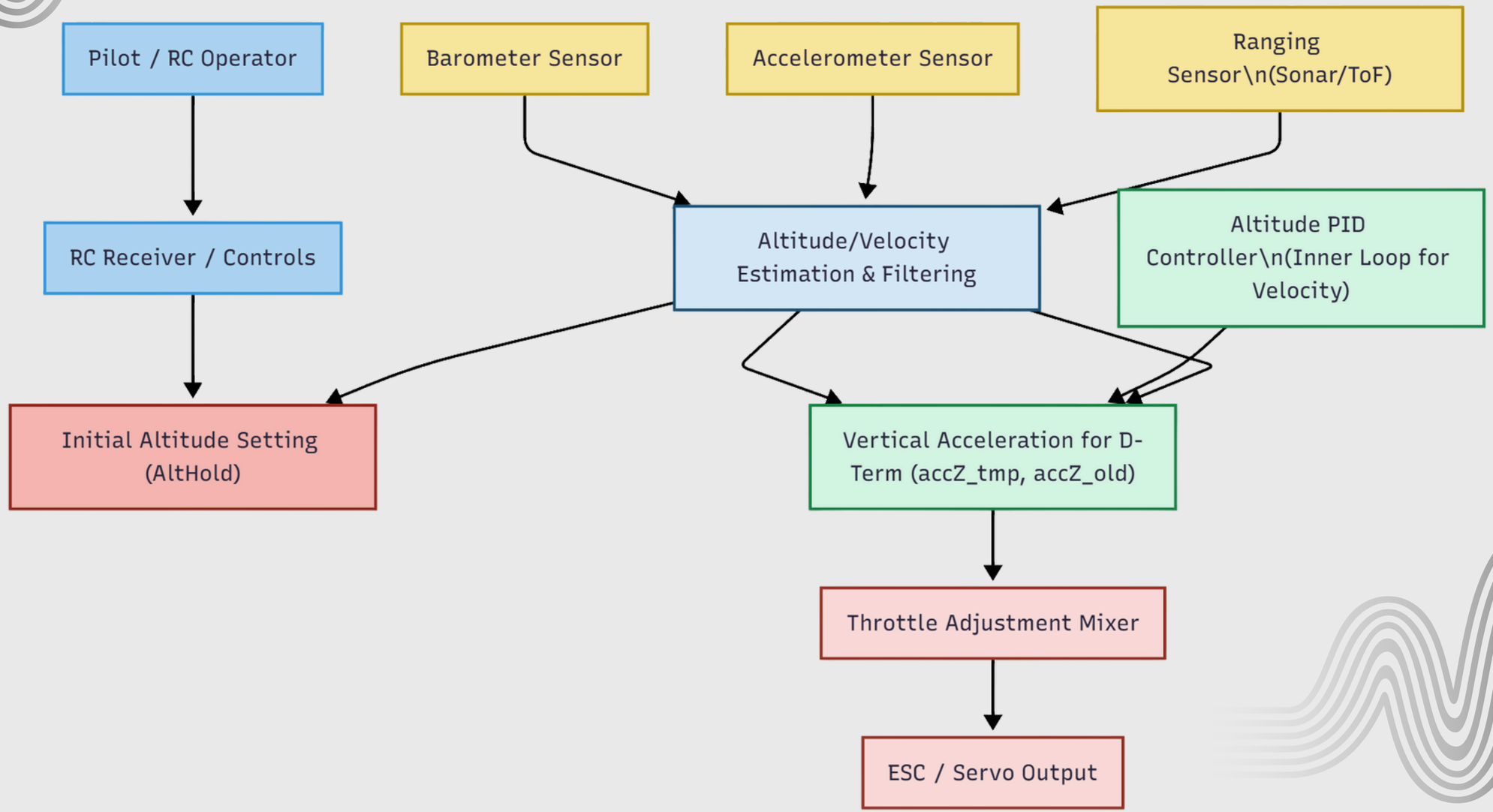
- To achieve altitude hold by using fusion of barometer and ToF sensor.
- To achieve minimal horizontal drift using IMU and Optical Flow sensor.
- To apply control logic to integrate the new sensors.
- To achieve 10-20 cm micro movements on user command.

SYSTEM ARCHITECTURE

1. Altitude Hold Architecture
2. Optical Flow Architecture
3. Overall Architecture

SYSTEM ARCHITECTURE

Altitude Hold Architecture



SYSTEM ARCHITECTURE

Altitude Hold Architecture

Altitude hold uses the barometer, accelerometer, and ranging sensor. The barometer and ranging sensor give continuous height data, which altitudehold.cpp fuses to estimate altitude (EstAlt) using the ranging_vl53l1x.cpp driver.

Why a ranging sensor?

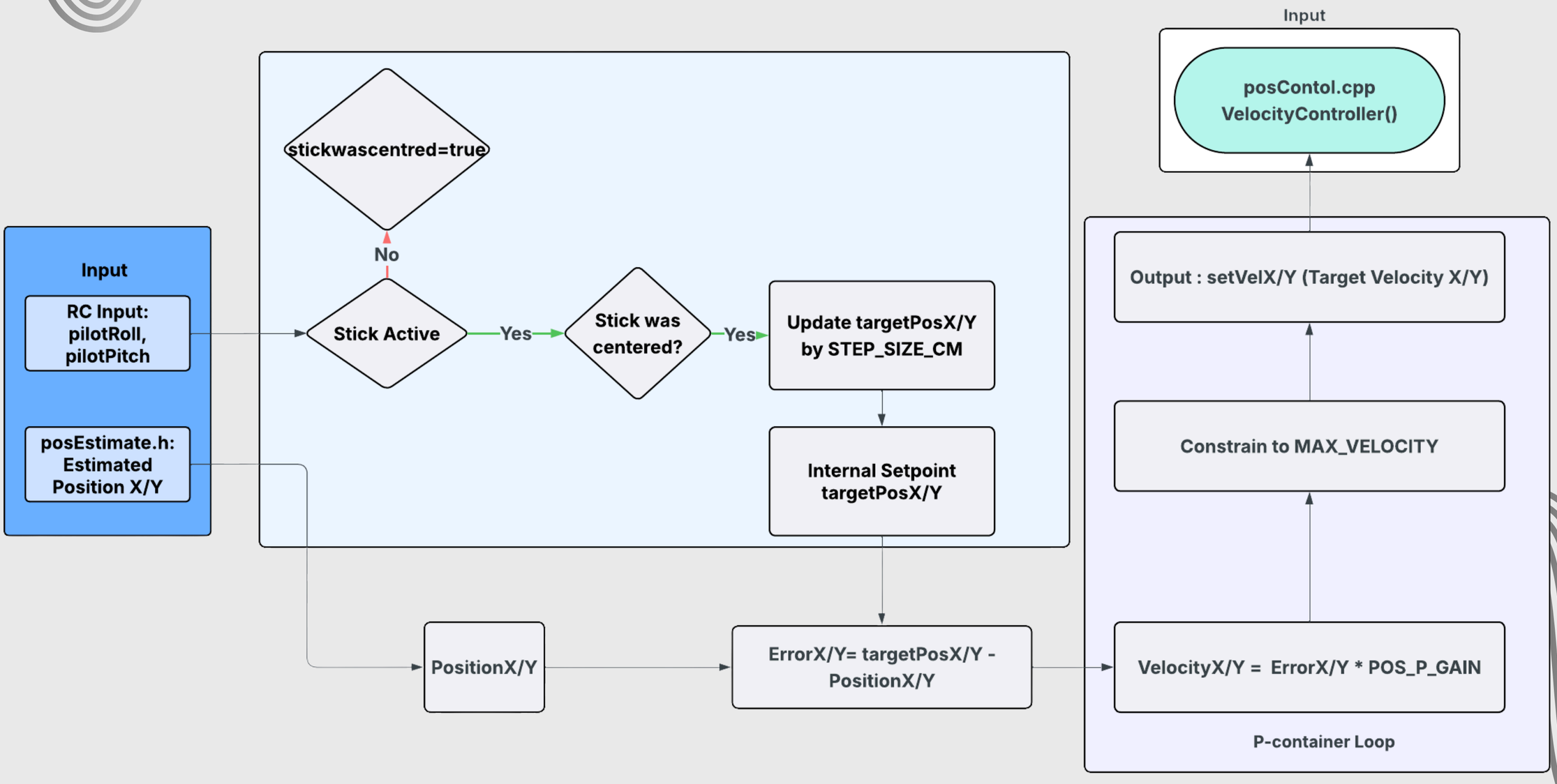
Barometer readings change with environmental conditions like atmospheric pressure. The ranging sensor gives stable, unaffected distance data, and combining both improves height accuracy.

Inner-loop integration

EstAlt is combined with vertical acceleration from the accelerometer for the D-term of the inner PID loop, reducing vertical velocity to zero and holding the altitude set by the RC receiver.

SYSTEM ARCHITECTURE

Optical Flow Architecture



SYSTEM ARCHITECTURE

Optical Flow Architecture

RC input (roll/pitch) and the estimated XY position from posEstimate.h are given to opticalflow.cpp. The stick management function decides horizontal movement or position hold: a non-centered stick commands movement, while a centered stick triggers position hold.

Estimated PositionX/Y from posEstimate is combined with the target setpoint in the control loop to calculate required horizontal velocity, which is sent to posControl.cpp for accurate control.

Why Optical Flow?

IMUs face noise from vibrations and EM interference and also drift. GPS can correct this, but mini drones cannot use GPS, so optical flow sensors correct these errors by estimating velocity from frame changes.

SYSTEM ARCHITECTURE

Integration of Optical Flow sensor in the inner loop

The legacy driver is replaced with the new PAW3903 driver (opticflow.cpp) to read sensor data. It converts the data into body rate and frame rate, applies adjustments for vibrations and tilt, and then computes deltaX and deltaY, which are passed to opticalflow.cpp.

The calculatesensorflow() function uses deltaX and deltaY and, after applying time- and height-based calculations, produces the corrected horizontal velocity.

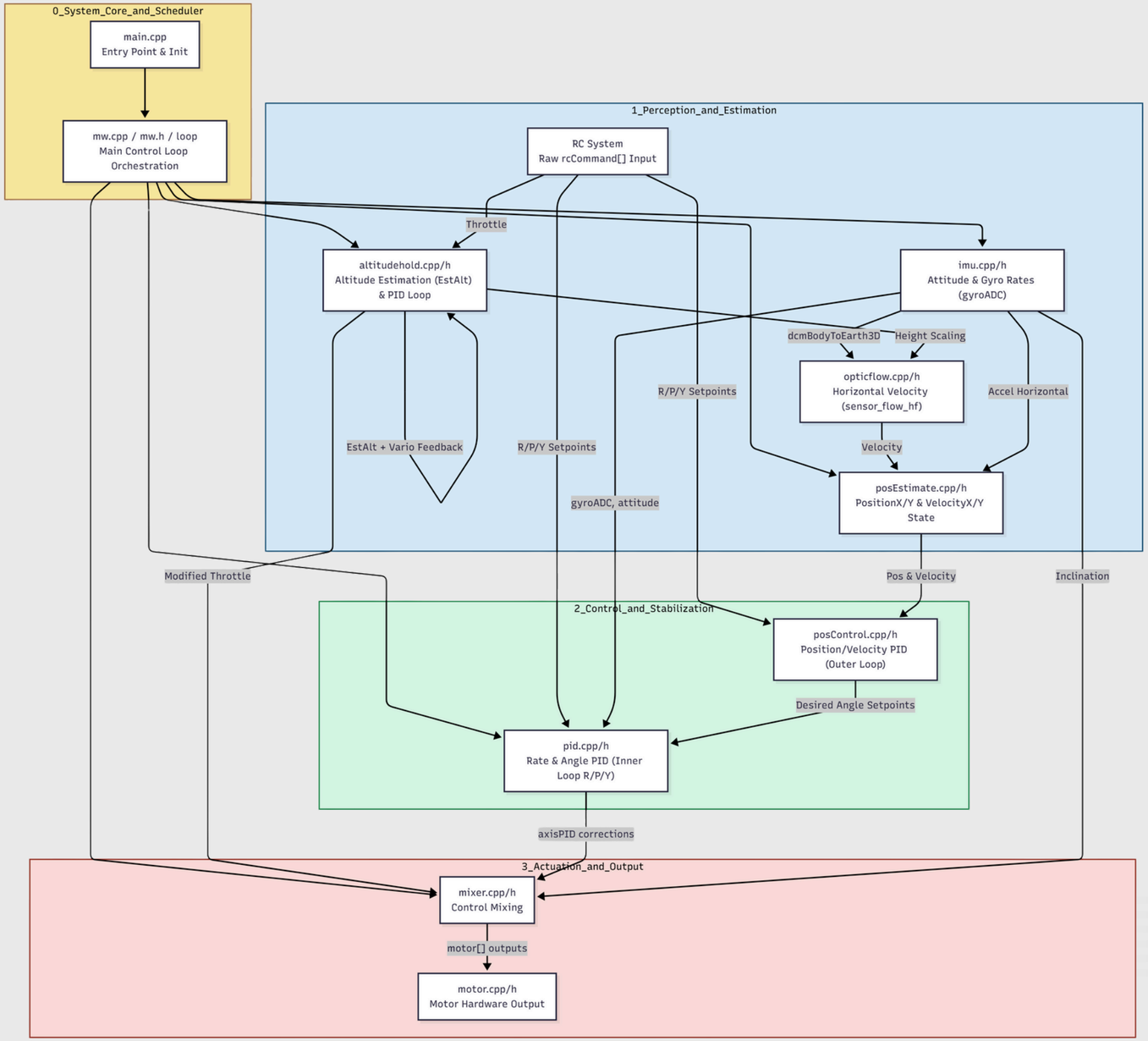
Together, the horizontal velocity outputs from opticflow.cpp and the vertical throttle adjustments from altitudehold.cpp according to the rcCommand[Roll/Pitch/Altiltude] feed into the mixer.cpp stage, producing the final motor PWM signals for flight.

SYSTEM ARCHITECTURE

```
// 2. READ SENSOR
PAW3903_Data flowData;
if (!paw3903_read_motion_burst(flowData)) {
    return;
}

// 3. CHECK QUALITY
if (flowData.squal < 15) {
    return;
}

// 4. GET RAW PIXELS
float raw_dx = (float)flowData.deltaX;
float raw_dy = (float)flowData.deltaY;

// 5. APPLY LOW PASS FILTER
filtered_raw_flow[0] = (filtered_raw_flow[0] * (1.0f - SENSORFLOW_LPS)) + (raw_dx * SENSORFLOW_LPS)
filtered_raw_flow[1] = (filtered_raw_flow[1] * (1.0f - SENSORFLOW_LPS)) + (raw_dy * SENSORFLOW_LPS)

// 6. GET ALTITUDE
float altitude_m = getLaserAltitudeMeters();
opticflowHeight = altitude_m;

// 7. CALCULATE VELOCITY (Using deltaT)
float vel_x_est = (filtered_raw_flow[0] * altitude_m * FLOW_SCALER) / deltaT;
float vel_y_est = (filtered_raw_flow[1] * altitude_m * FLOW_SCALER) / deltaT;
```

SYSTEM ARCHITECTURE

Overall Architecture



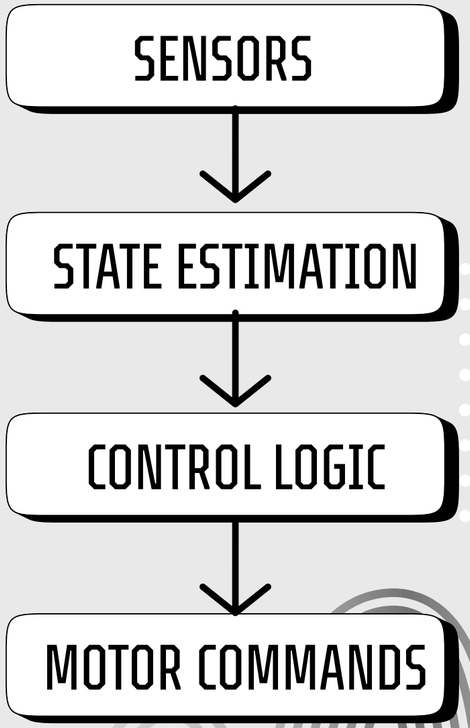
SYSTEM ARCHITECTURE

System Core
main.cpp initializes all drivers and then transfers control to mw.cpp, which functions as the Scheduler. It manages the synchronized main control loop for sensor reading, state estimation, and control logic.

Perception and Estimation
The system processes raw RC inputs and sensor data:
- altitudehold.cpp fuses barometer and ToF readings to compute estimated altitude (EstAlt).
- imu.cpp provides gyro rates.
- opticflow.cpp computes horizontal velocity, which posEstimate.cpp uses to track Position X and Y.

Control and Stabilization
State estimates feed into a cascaded control structure:
- posControl.cpp forms the Outer Loop, generating required angle setpoints from estimated position.
- These setpoints are passed to pid.cpp, the Inner Loop, which computes rate and angle PID corrections.

Actuation and Output
mixer.cpp combines the altitude controller's throttle output with the rate controller's PID corrections to generate final motor PWM signals, which motor.cpp sends to the hardware.



SENSOR FUSION

Altitude Hold System (Tof Sensor and Barometer fusion)

Fusion Hierarchy

The system prioritizes the VL53L1X Time-of-Flight (ToF) sensor as the primary height reference because it provides accurate measurements in most conditions. The algorithm first validates the ToF reading (expected between 5 cm and 3.5 m). If valid, it is used directly. If the drone exceeds this range or the sensor becomes unreliable, the system automatically switches to the barometer to ensure continuous altitude estimation.

Tilt Compensation (Geometric Correction)

What is the need of geometric compensation?

Because the ToF sensor is rigidly mounted to the drone's body, any pitch or roll during flight causes it to measure a slant distance instead of the true vertical height. To correct this, the measured distance is multiplied by the cosine of the tilt angle, restoring the actual altitude.

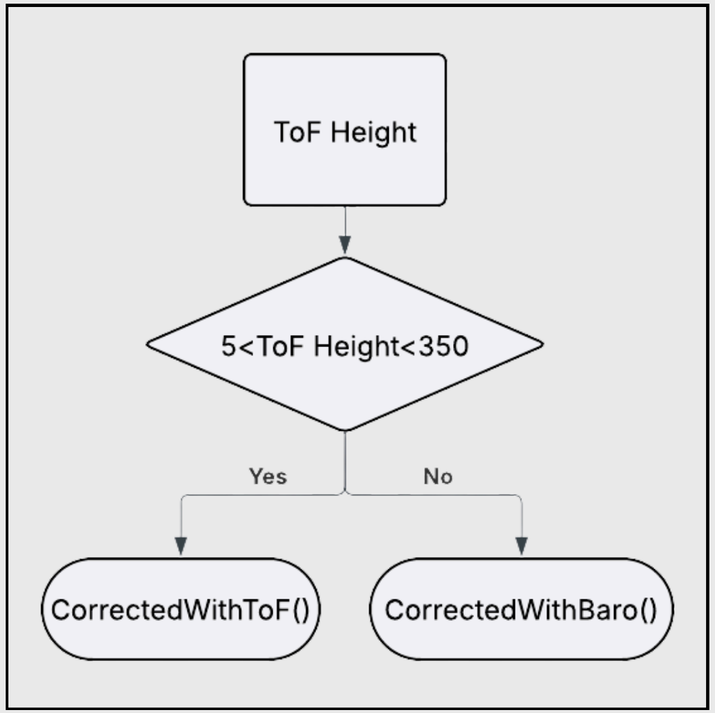
Filtering (Complementary + Kalman)

To reduce sensor noise, a third-order complementary filter fuses the accelerometer's fast response with the long-term stability of the position sensors. The resulting vertical velocity is then refined using a 1-D Kalman Filter, which further smooths the Velocity-Z estimate before it enters the PID controller.

SENSOR FUSION

```
if ( isTofDataNew_L1 () && ! isOutOfRange_L1 () ) {
    ToF_Height = ( float ) NewSensorRange_L1 / 10.0 f ;
    tilt = degreesToRadians ( calculateTiltAngle (& inclination )
    if ( tilt < 25.0 f )
    ToF_Height *= cos_approx ( tilt ) ;
}
```

```
if ( ToF_Height > 5.0 f && ToF_Height < 350.0 f && ! isOutofRange_L1 () )
{
    baro_offset = filtered - EstAlt ;
    correctedWithTof ( ToF_Height ) ;
} else {
    correctedWithBaro ( Baro_Height - baro_offset , dt ) ;
}
```



TELEMETRY

```
Monitor_Println("GRAPH:alt_est_cm",  (double)EstAlt);
Monitor_Println("GRAPH:vel_z",       (double)VelocityZ);
Monitor_Println("GRAPH:tof_height_cm", (double)ToF_Height);
```

THIS ASSISTS IN TUNING COMPLEMENTARY-FILTER GAINS AND KALMAN PARAMETERS.

SENSOR FUSION

Optical Flow System (PAW3903 + ToF-Based Horizontal Velocity)

Velocity Scaling

For horizontal stability, raw optical flow pixels are not enough, they have no sense of scale. A pixel moving at 1 meter height represents a different speed than a pixel moving at 2 meters. We solve this by fusing the flow data with our ToF Altitude estimate. The algorithm computes real world velocity by multiplying the filtered flow rate by the current height:

```
vel_x_est = (filtered_flow[0] * altitude * FLOW_SCALER) / deltaT;
vel_y_est = (filtered_flow[1] * altitude * FLOW_SCALER) / deltaT;
```

Where, filtered flow is the low pass filtered pixel delta.

Telemetry

- Estimated velocities:

VelX Bench, VelY Bench (continuous cm/s)

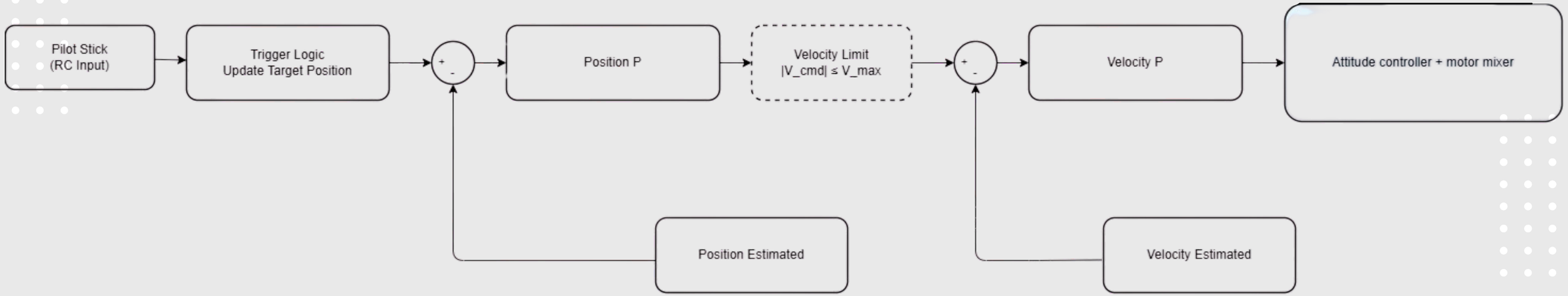
- Raw sensor data:

FlowX Raw, FlowY Raw (pixel deltas)

Squal (surface quality)

CONTROL STRATEGY

Horizontal control uses a cascaded P–P controller for both X and Y axes.



CONTROL STRATEGY

Outer Loop (Position Controller)

Position error at time t,

$$e_p(t) = P_{target} - P_{current}$$

Desired Velocity,

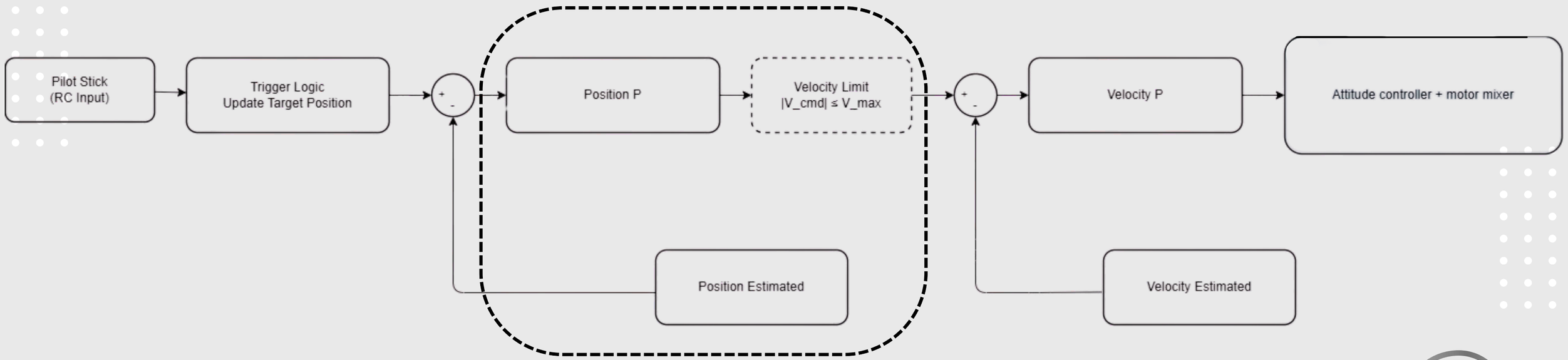
$$V_{cmd} = K_{p\_pos} \cdot e_p(t)$$

(We utilize a P-only controller for the outer loop because the inner velocity loop naturally acts as a damping factor, preventing overshoot)

Then, constrained to a safe maximum velocity:

$$V_{cmd} = \text{clamp}(V_{cmd}, -V_{\max}, V_{\max})$$

OUTER LOOP



OUTER LOOP

CONTROL ARCHITECTURE STRATEGY

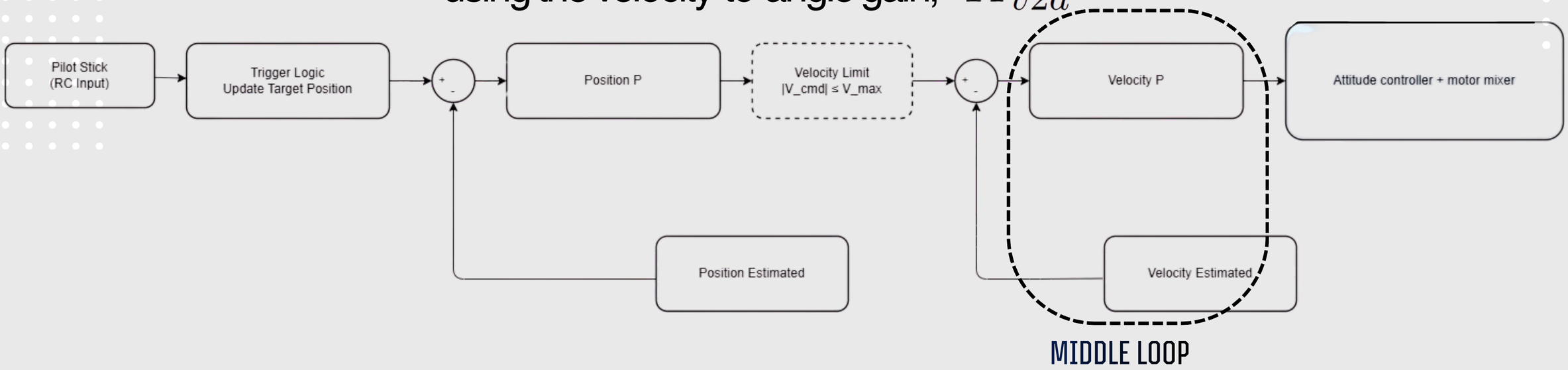
Middle Loop (Velocity Controller)

Velocity command is converted to an attitude (tilt) command:

$$e_v(t) = V_{\text{cmd}}(t) - V_{\text{est}}(t)$$

$$\theta_{\text{cmd}} = K_{v2a} \cdot e_v(t)$$

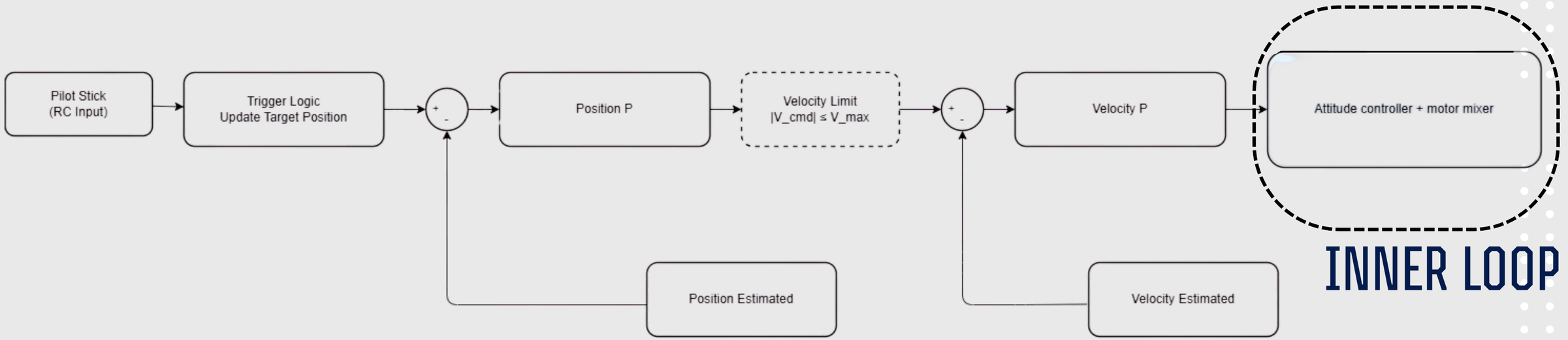
using the velocity-to-angle gain, K_{v2a}



CONTROL ARCHITECTURE STRATEGY

Inner Loop (Attitude Controller)

The final tilt commands are passed to the built-in high-rate attitude PID loop of the Magis firmware. This loop handles fast stabilization by tightly controlling roll, pitch, and yaw to track the commanded angles.



MICROMOVEMENT LOGIC

A **Latched Trigger mechanism** is used .
The drone doesn't move based on how long you hold the stick, but on the event of the stick crossing a threshold.

A flag stickWasCentered ensures the drone executes only one single step before forcing the pilot to release the stick back to center. This guarantees compliance with the discrete movement requirement and prevents runaway motion.

When the input exceeds a safety dead band, the global positional setpoint is instantaneously updated by a fixed quantum:

$$P_{newTarget} = P_{oldTarget} \pm STEP\_SIZE\_CM$$

where ***STEP_SIZE_CM*** *is fixed at* ***15 cm***.

TUNING STRATEGY

Z-Axis Tuning

- Tuned the Laser Time Constant to **1.5s** for rapid sensor fusion response.
- Horizontal optical flow calculations depend on accurate altitude ($V \propto h$). If height wobbles, velocity estimates become garbage.

Sensor Normalization:

- **I**nitial bench tests showed unrealistic velocity spikes from minor hand movements.
- Calibrated the Flow Scalar down to **0.001**.

TUNING STRATEGY

Horizontal Loop Tuning (Inside-Out):

- Inner Loop First ($V \rightarrow \theta$): Tuned **VEL_TO_ANGLE** gain to **3.0**. This provided Active Damping, ensuring the drone brakes automatically when moving too fast.

- Outer Loop Second ($P \rightarrow V$): Tuned **POS_P_GAIN** to **1.5**. This set the aggression for returning to the target position without overshooting.

EXPERIMENTS & RESULTS

Using the fused ToF and barometer estimate,

- the drone climbed smoothly to around 1.5 meter and held its height within the 0.5–2 meter band with very small fluctuations.

Altitude error

Vertical Velocity

Throttle Cascade

- For the optical flow module there was a small drift in position over a hover test.

then, we gave roll, the drone moved in that direction and then stabilized after the stick is centered.

- FLOW_SCALER TUNING
- DEADBANDING
- SQUAL GATING

FAILURE HANDLING

We identified three major risks:

- velocity saturation(clamp maximum velocity commands)
- corrupted optical-flow frames(reject low-quality frames using the SQUAL threshold)
- loss of ToF altitude(fall back to barometer data whenever the ToF becomes unreliable) >3.5m

LIMITATIONS AND IMPROVEMENTS

- The main limitation right now is optical-flow asymmetry and noise sensitivity, especially on the Y-axis
- the logic still depends heavily on raw pixel motion, some slow drift can appear under challenging textures or lighting.
- A PI-based velocity controller and an automatic descent failsafe during prolonged low SQUAL would significantly improve consistency in future iterations.

THANK YOU

by Team 24