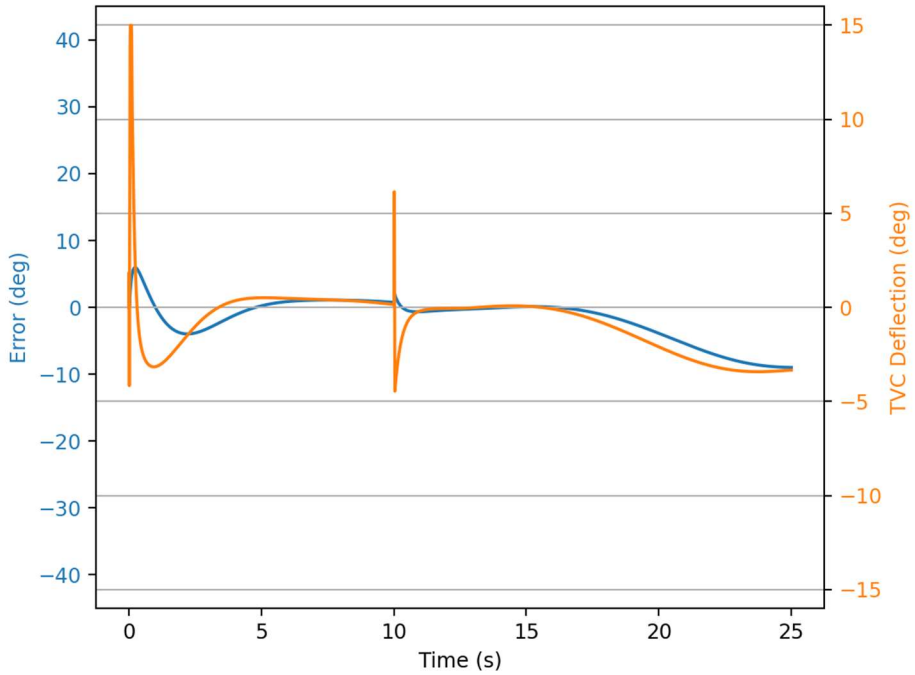
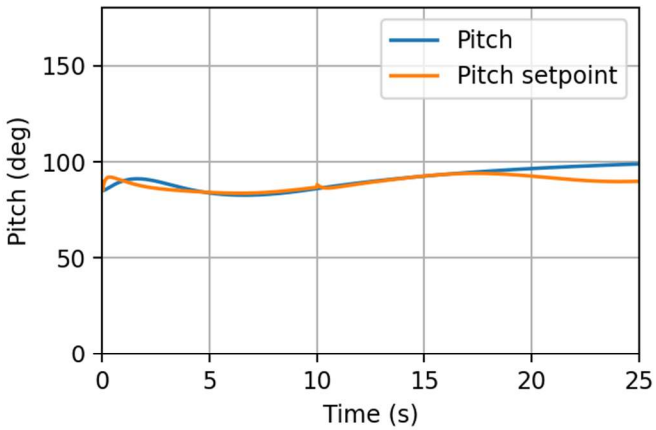
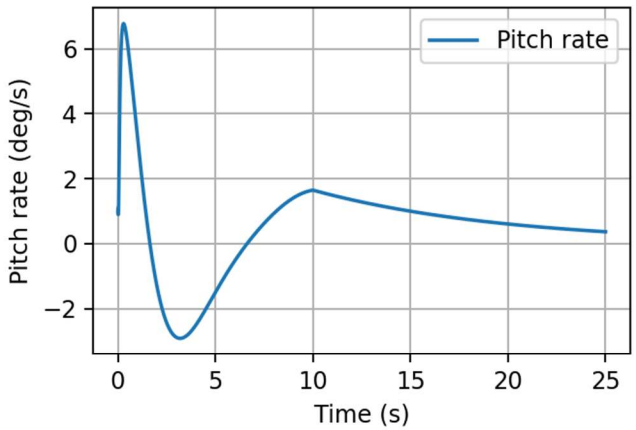
Development process

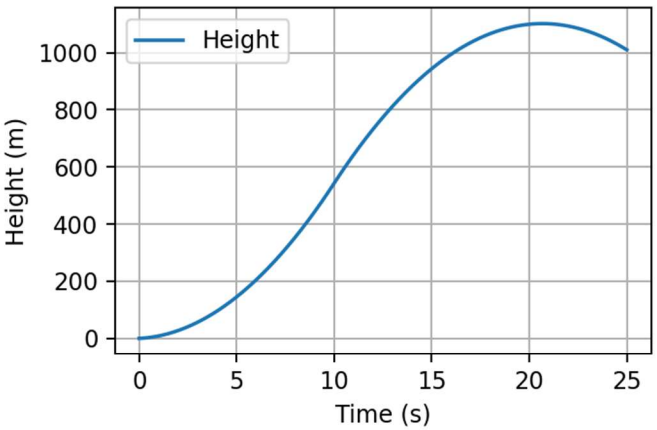
2. Simulation

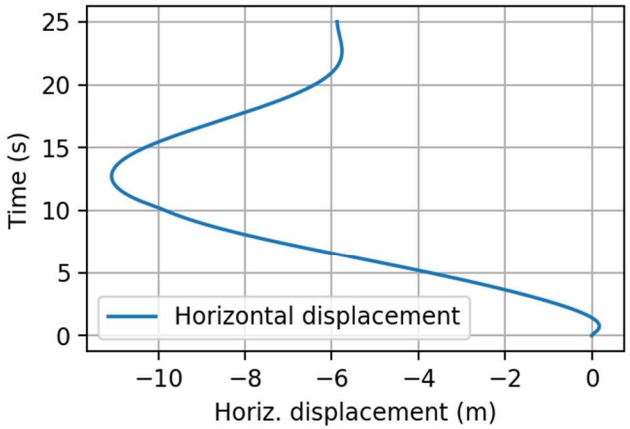
- Written in Python
- 2D 3-DOF (x/y, rotation)
  - Rigid body simulation
- Test and verify physics and PID controller
  - 1-axis pitch
  - 1-axis horizontal drift due to wind
- Determining moment of inertia
  - Bifilar pendulum method

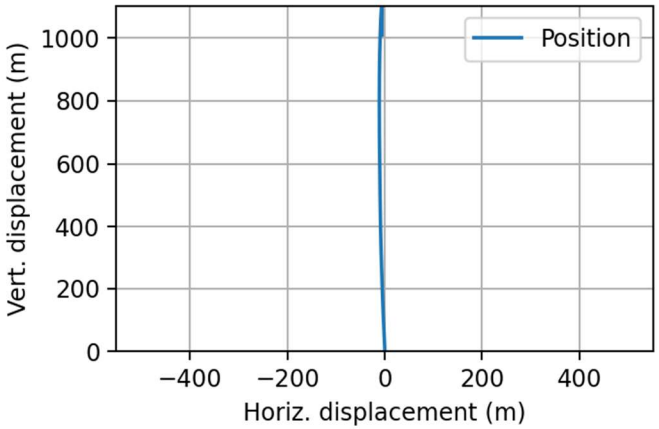


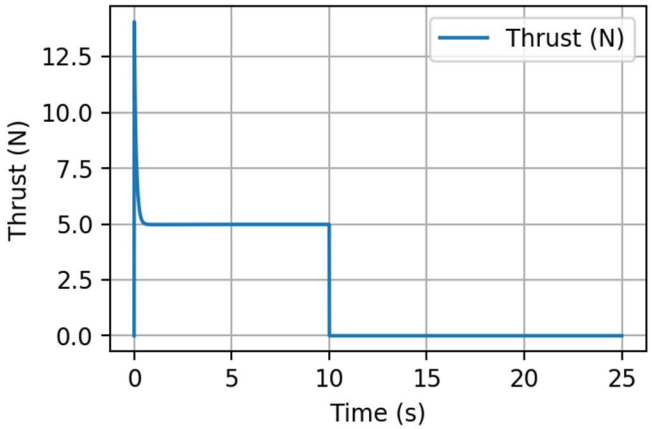

Pitch
Pitch setpoint
Pitch (deg)
Time (s)


Pitch rate
Pitch rate (deg/s)
Time (s)


Height
Height (m)
Time (s)

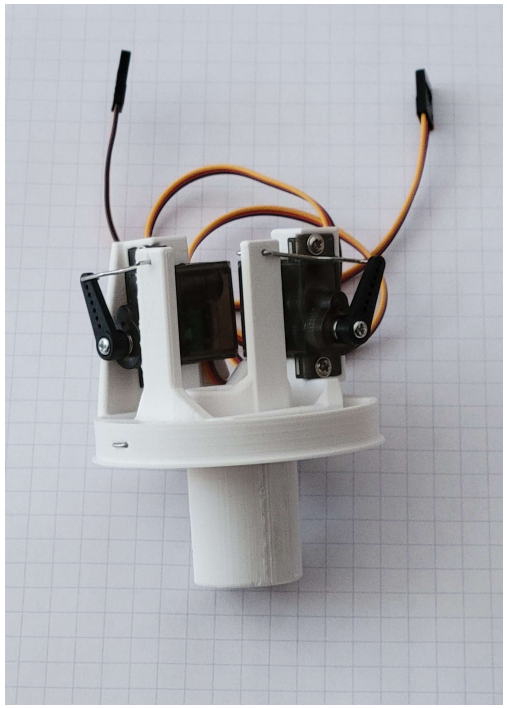

Horizontal displacement
Time (s)
Horiz. displacement (m)

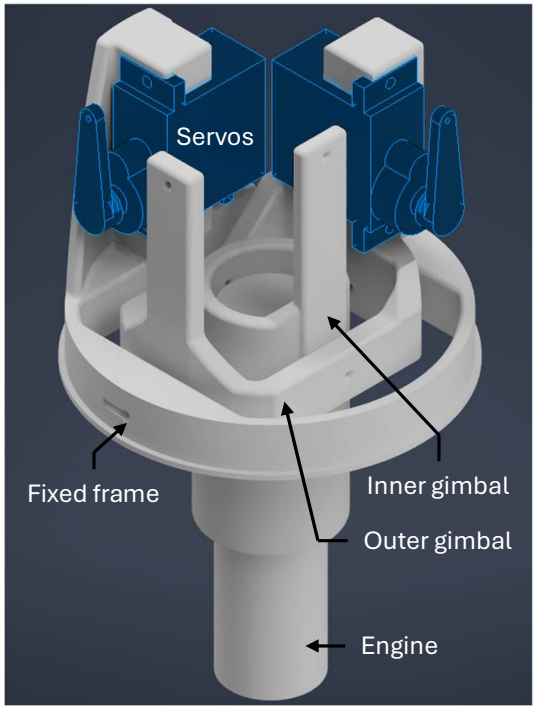

Position
Vert. displacement (m)
Horiz. displacement (m)

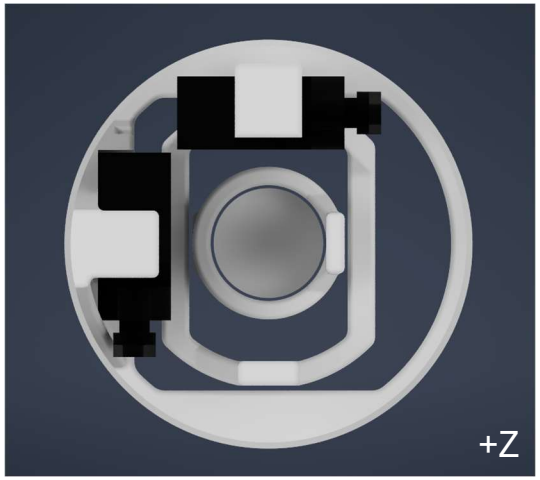

Thrust (N)
Thrust (N)
Time (s)

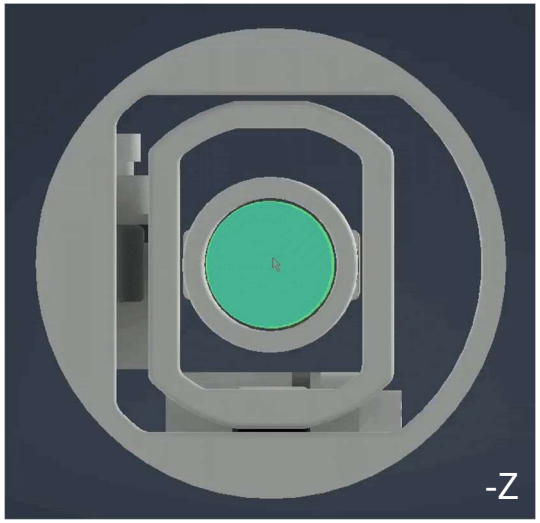
3. Mechanism design

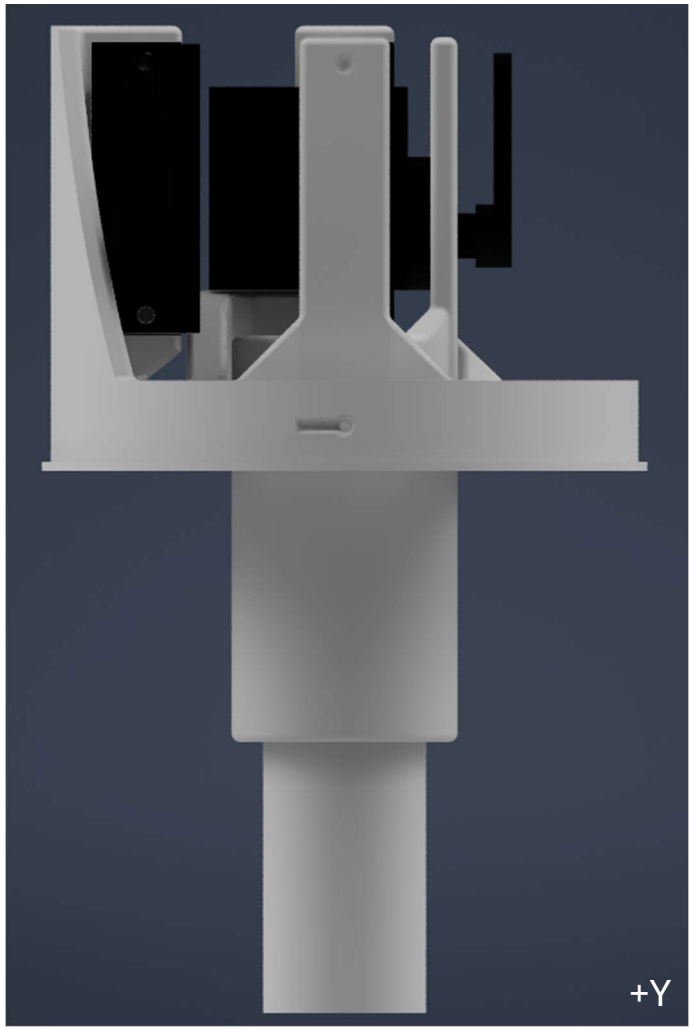
- 2-axis gimbal
- Servo mounts
- Actuation levers
- 3D printed in PETG
  - Durability
  - Flexibility
  - Heat resistance
- Bearings and connectors
  - Paperclip wire

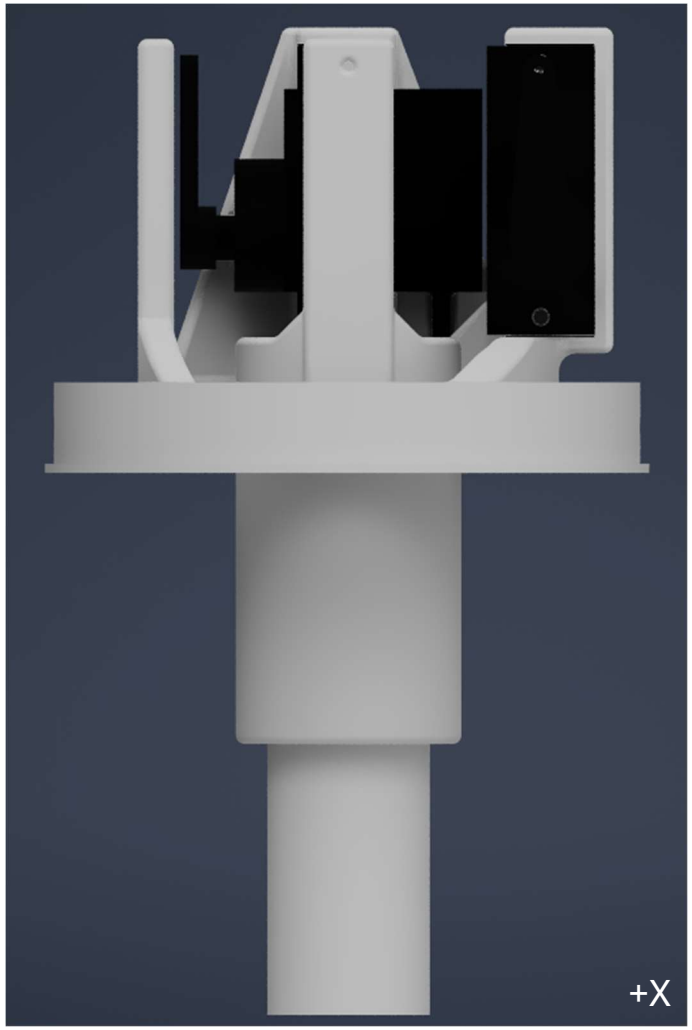





+Z

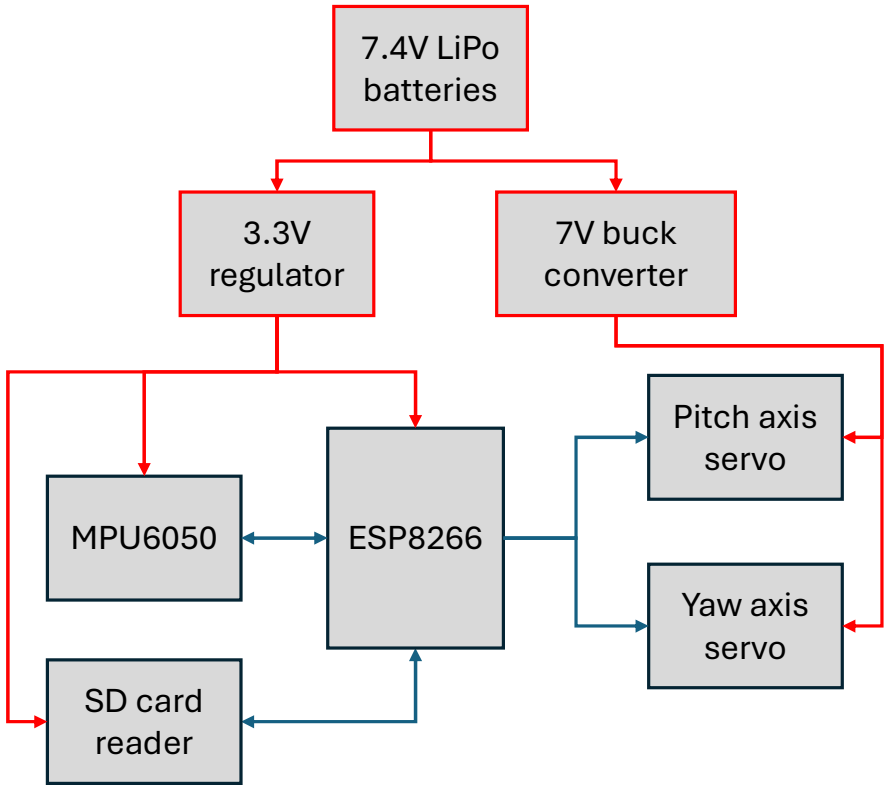

-Z


+Y


+X

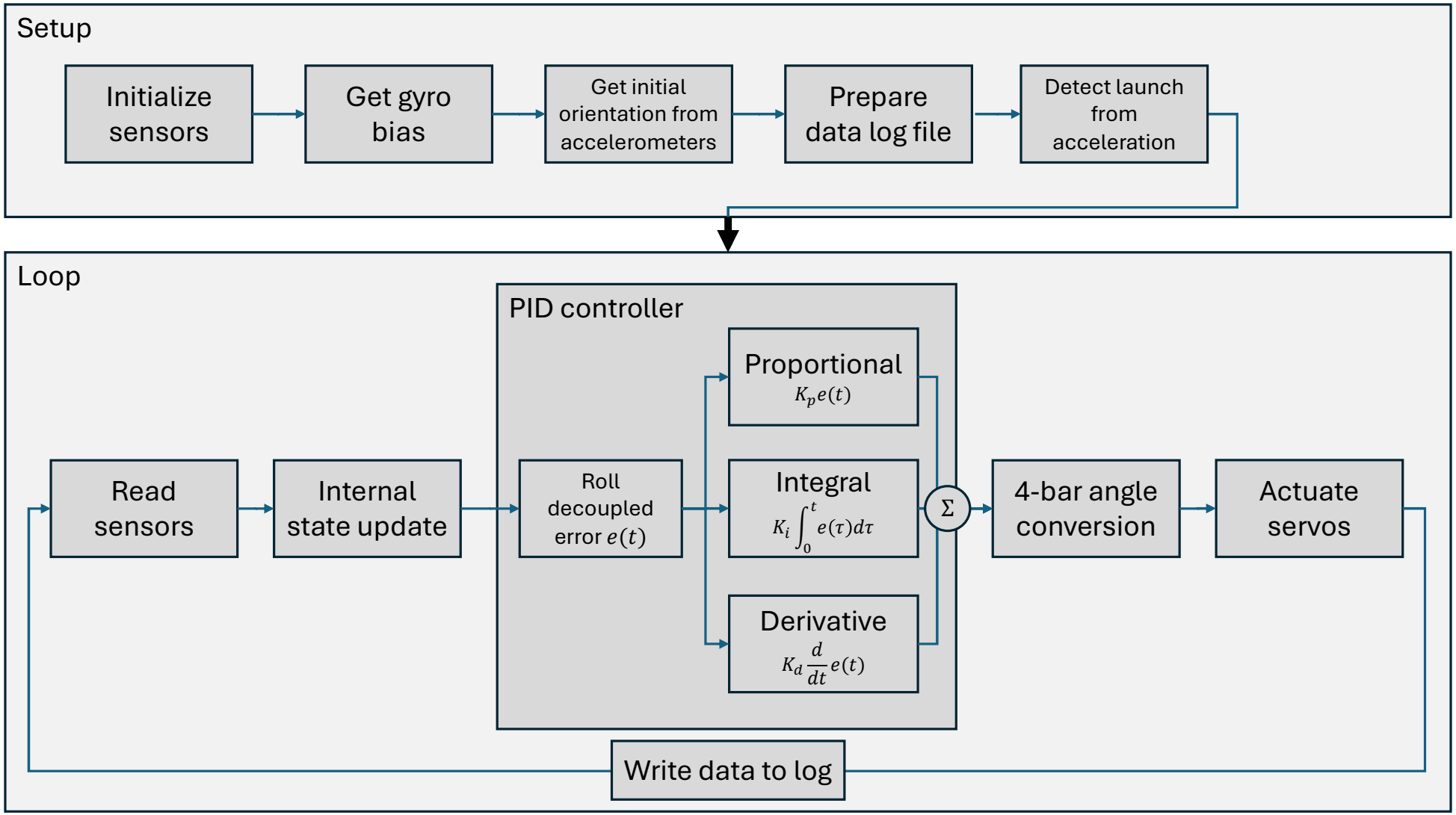
4. Electronics

- ESP8266 microcontroller
  - Real-time processing
  - Wireless connectivity
  - Lightweight
- MPU6050 IMU sensor
  - Rate gyro and accelerometer
- MG90S metal-geared servos
  - Durability and torque
- SD card reader for data logging



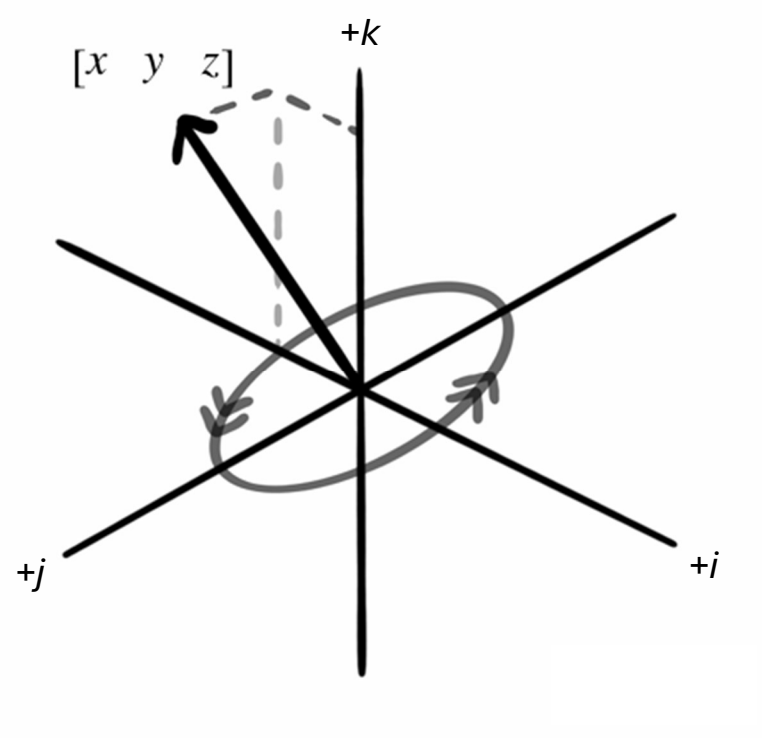
5. Control software design

- Written in C++
  - Arduino and sensor libraries
  - Custom vector and quaternion math library
- Calibrate sensors
- Track current rocket state
  - Integrate angular rate and acceleration
- Process sensor data, compensate for errors
- Calculate required correction
- Calculate servo angles and execute


Setup
Initialize sensors
Get gyro bias
Get initial orientation from accelerometers
Prepare data log file
Detect launch from acceleration
Loop
PID controller
Proportional
$K_p e(t)$
Integral
$K_i \int_0^t e(\tau)d\tau$
Derivative
$K_d \frac{d}{dt} e(t)$
Σ
Read sensors
Internal state update
Roll decoupled error $e(t)$
4-bar angle conversion
Actuate servos
Write data to log

5.0. Quaternions and 3D rotations

- 4D unit vector (real, *i, j, k*) of length 1
- Used to represent 3D rotations
  - Multiplication (Hamilton product) composes rotations
- Avoids problems with Euler angles
  - Prevents gimbal lock (2 axes align)
  - Better interpolation between angles
  - Eliminates ambiguity with rotation orders
  - More efficient computation



5.1. Updating internal state

- Get initial orientation from gravity vector
- Local to world frame transform quaternion
  - Integrate quaternion derivative from rate gyro

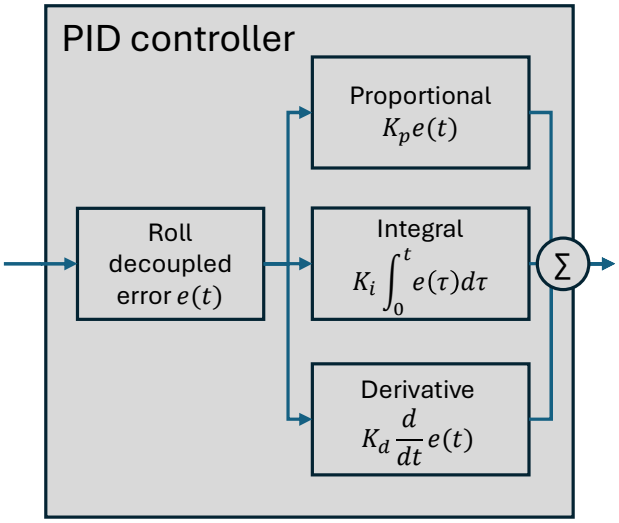
$$dq = \frac{1}{2}\left[\frac{2}{dt}, \omega_x, \omega_y, \omega_z\right] dt \quad Q_{i+1} = Q_i \otimes \frac{dq}{\|dq\|}$$

- Decouple pitch and yaw from roll axis
  - Convert Q to Euler angles (pitch, yaw, roll)
  - Isolate roll axis in a separate quaternion Q_{roll}
  - Remove roll: ${Q_{rol}}^{-1} \otimes Q$

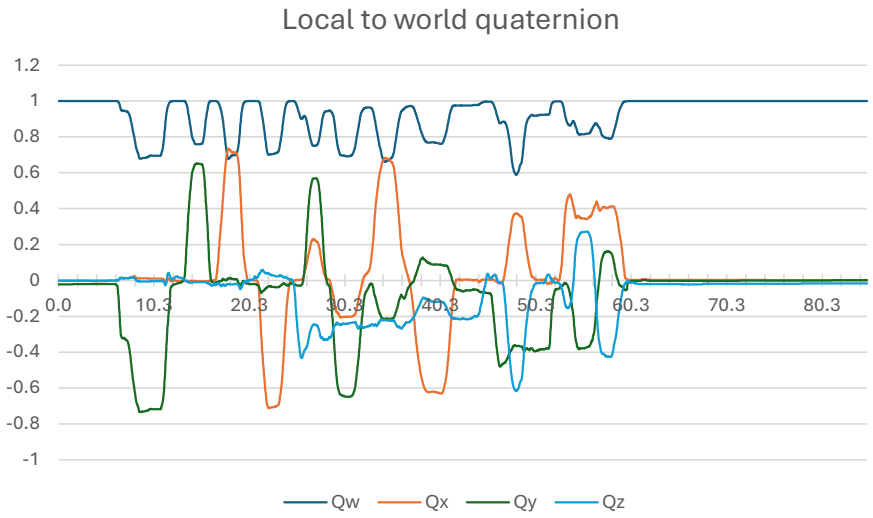
```
/**
 * Updates the current physical state of the rocket
 */
void updateState()
{
  // Update sensor readings
  mpu.getEvent(&a, &g, &temp);

  // --- Update rotation ---
  // Get current angular velocity and prepare for quaternion derivative
  omega = (Vector3d::fromSensorData(g.gyro) - gyroErr) * 0.5 * dt;
  // Integrate by the quaternion derivative
  QuaternionD dq = QuaternionD(1, omega.x, omega.y, omega.z).normalized();
  // globalRot * dq → body to world, dq * globalRot → world to body
  globalRot = (globalRot * dq).normalized();
  // Convert to world frame → local frame Euler angles with roll decoupled
  eulerRot = (QuaternionD::fromEulerAngles(
                0, 0,
                (globalRot.conjugate().toEulerAngles()).z).conjugate() * globalRot
              ).toEulerAngles();
  // --- Decouple rotation axes with z=roll ---
  decoupledRot = Vector2d(eulerRot.x, eulerRot.y);
}
```

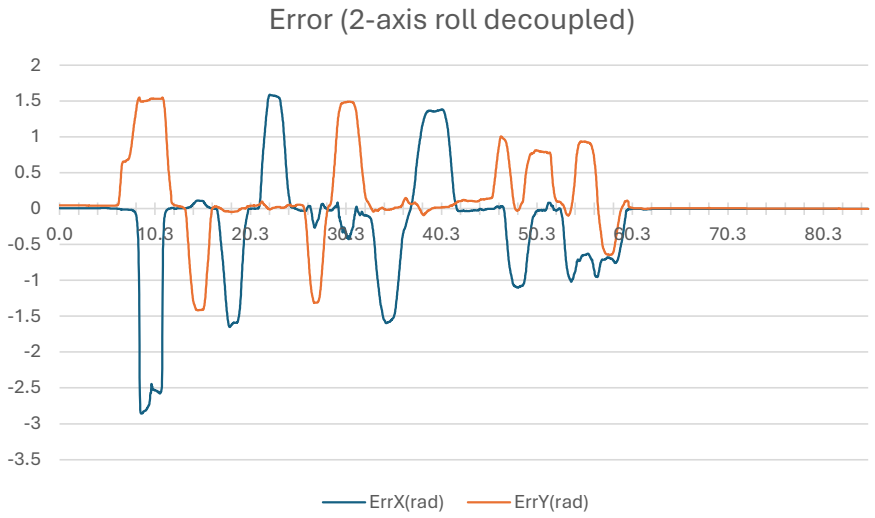
5.2. PID controller (axis-based)

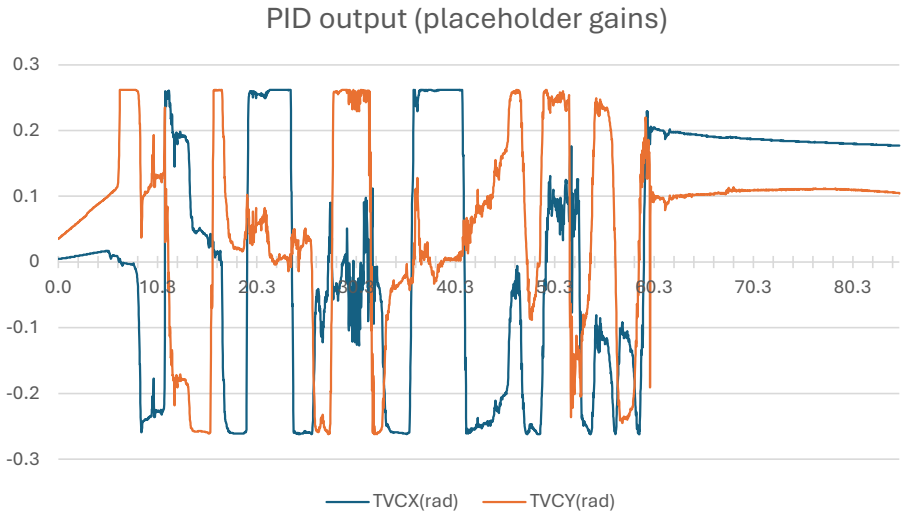
- Attempts to minimize calculated error
- Error = setpoint – position
- Proportional term
  - Directly corrects error
- Derivative term
  - Rate of change of error
- Integral term
  - Accumulate steady state error
  - Anti-windup
- Overall correction = sum of each term




```
/**
 * Pair of one-dimensional PIDs
 * @param setpoint Setpoint in pitch and yaw angles
 * @param position Current position in pitch and yaw angles
 * @returns Correction angle in pitch and yaw in the same units
 */
Vector2d AxesPID(Vector2d setpoint, Vector2d position)
{
  Vector2d newError = setpoint - position;
  Vector2d p = newError * KP;
  PIDInt += newError * KI * dt;
  Vector2d d = (newError - PIDErr) * KD / dt;
  PIDErr = newError;
  return p + PIDInt + d;
}
```


Local to world quaternion
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0.0
10.3
20.3
30.3
40.3
50.3
60.3
70.3
80.3
Qw
Qx
Qy
Qz


Error (2-axis roll decoupled)
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
0.0
10.3
20.3
30.3
40.3
50.3
60.3
70.3
80.3
ErrX(rad)
ErrY(rad)


PID output (placeholder gains)
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
0.0
10.3
20.3
30.3
40.3
50.3
60.3
70.3
80.3
TVCX(rad)
TVCY(rad)

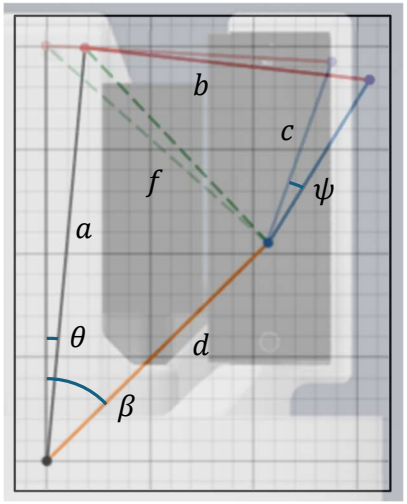
5.3. Gimbaling and servo actuation

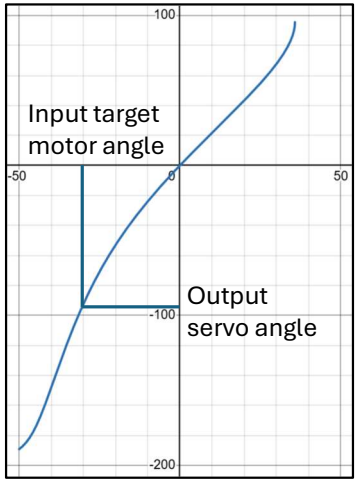
- 4-bar linkage
- Convert target angle θ to servo angle ψ
  - Equations derived using trig

$$f(\theta) = \sqrt{a^2 + b^2 - 2ad\cos\theta}, \qquad g(\theta) = \operatorname{asin}\left(\frac{a\sin(\theta)}{f(\theta)}\right),$$

$$h(x) = -\operatorname{acos}\left(\frac{b^2 - f(x)^2 - c^2}{-2f(x)c}\right) - \begin{cases} g(x), & |x| \le \operatorname{acos}\left(\frac{d}{a}\right) \\ \pi\,\operatorname{sign}(x) - g(x) \end{cases}$$

$$\boxed{\psi(\theta) = h(-\theta - \beta) - h(\beta)}$$






```
/**
 * Converts target output angle to required servo angle using 4-bar linkage
 * @param trad Target output angle in rad.
 * @param ax Dimensions and parameters of the 4-bar linkage
 * @returns Servo angle in deg. required for the target angle
 */
double getServoAngleFromTargetAngle(double trad, FourBarParams ax)
{
  trad = -trad - ax.n;
  double f = sqrt(ax.aSqr + ax.dSqr - ax.ad2 * cos(trad));
  double as = asin((ax.a * sin(trad)) / f);
  double a = abs(trad) <= ax.acosLim ? f : copysign(PI, trad) - f;
  double c = acos((ax.bSqr - f * f - ax.cSqr) / (-2 * f * ax.c));
  return ax.zero - degrees(a + c);
}
```