Implementation and Control of an Underactuated Ballbot System

Technical Presentation

Robotics Engineering

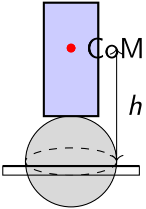
February 11, 2026

Outline

1. Introduction to Ballbot Concept
2. Mechanical Architecture
3. Hardware Architecture
4. System Modeling
5. Dynamic Equations
6. Omni Kinematics
7. Control System Design
8. Software Architecture
9. Power and Driver Calculations
10. Stability and Tuning
11. Limitations and Extensions
12. Conclusion

What is a Ballbot?

- Dynamically stable robot
- Balances on a single spherical ball
- Inherently unstable (inverted pendulum)
- Omnidirectional mobility
- Underactuated MIMO system



Key Challenges

Underactuated System

3 actuators (motors) controlling 5 degrees of freedom:

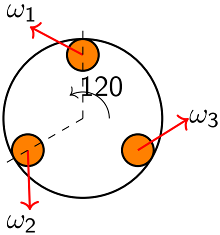
- Body tilt (2 DOF): θ_x, θ_y
- Body rotation (1 DOF): ψ
- Ball position (2 DOF): x, y

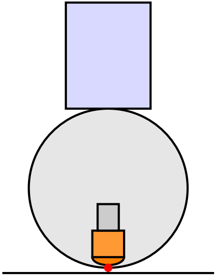
Control Requirements

- Fast sensor sampling ($>$ 100 Hz)
- Low-latency control loop
- Robust to disturbances
- Smooth omnidirectional motion

Three-Roller Internal Drive

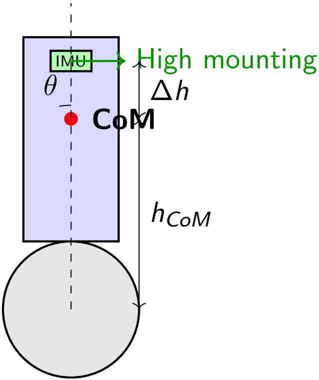
Top View (120° Configuration)





Side View

Center of Mass and IMU Placement

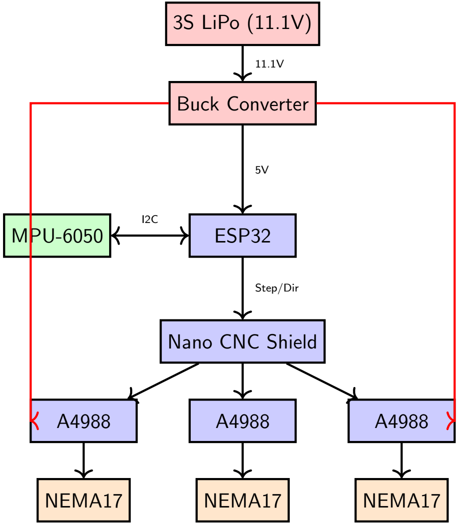


Design Criteria:

- CoM high above ball center
- Increased $h \to$ longer time constant
- IMU near top for better sensitivity
- Symmetric mass distribution

$$T_{fall} \propto \sqrt{\frac{h}{g}}$$

Electronics Block Diagram



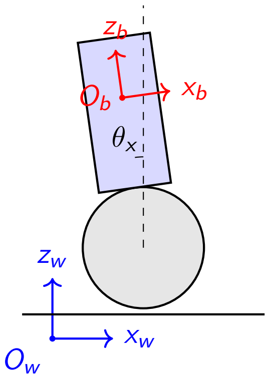
Component Specifications

Component	Specification	Purpose
ESP32	240 MHz dual-core	Control + sensor fusion
MPU-6050	6-axis IMU, 1 kHz max	Tilt measurement
A4988	2A max, 1/16 microstep	Stepper driver
NEMA17	1.5A, 0.4 Nm	Roller actuation
3S LiPo	11.1V, 2200 mAh	Power supply
Buck Conv.	11.1V $\rightarrow$ 5V, 3A	Voltage regulation

Key Design Choices

- ESP32: Sufficient speed for real-time control (¡10 ms loop)
- MPU-6050: Cost-effective 6-DOF with reasonable noise
- A4988: Simple step/dir interface, adequate current

Coordinate Frames



Frames:

- World frame: $\{O_w\}$
- Body frame: $\{O_b\}$
- Ball frame: $\{O_{ball}\}$

Tilt Angles:

$$\theta_x : \text{roll (about } x_w)$$
$$\theta_y : \text{pitch (about } y_w)$$

Small Angle Assumption:

$$|\theta_x|, |\theta_y| < 15$$

State and Input Vectors

State Vector

$$\boldsymbol{x} = \begin{bmatrix} \theta_x \\ \theta_y \\ \dot{\theta}_x \\ \dot{\theta}_y \end{bmatrix} \in \mathbb{R}^4$$

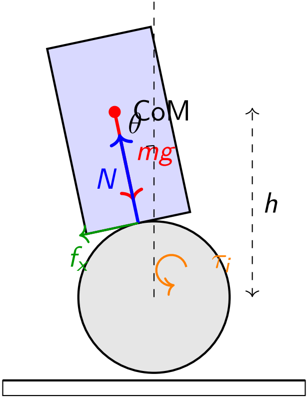
- θ_x, θ_y: Tilt angles (rad)
- $\dot{\theta}_x, \dot{\theta}_y$: Angular velocities (rad/s)

Input Vector

$$\boldsymbol{u} = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} \in \mathbb{R}^3$$

- τ_i: Torque from motor i (Nm)

Free Body Diagram



Inverted Pendulum Analogy

Linearized Dynamics:

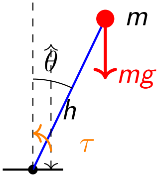
$$I\ddot{\theta} = mgh\sin\theta - \tau_{motor}$$
$$\approx mgh\theta - \tau_{motor}$$

For small angles:

$$\ddot{\theta} = \frac{mgh}{I}\theta - \frac{\tau_{motor}}{I}$$

Define: $\omega_n^2 = \frac{mgh}{I}$ (natural frequency)

$$\ddot{\theta} = \omega_n^2\theta - \frac{\tau_{motor}}{I}$$



Torque to Angular Acceleration

Motor torques produce body angular acceleration:

$$\begin{bmatrix} \ddot{\theta}_x \\ \ddot{\theta}_y \end{bmatrix} = \begin{bmatrix} \omega_n^2 & 0 \\ 0 & \omega_n^2 \end{bmatrix} \begin{bmatrix} \theta_x \\ \theta_y \end{bmatrix} + \frac{1}{I} \boldsymbol{J}_{motor} \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} \tag{1}$$

where $\boldsymbol{J}_{motor}$ maps motor torques to body angular accelerations.

For 120° configuration:

$$\boldsymbol{J}_{motor} = \frac{r_{ball}}{3} \begin{bmatrix} 0 & -\sqrt{3}/2 & \sqrt{3}/2 \\ 1 & -1/2 & -1/2 \end{bmatrix}$$

where r_{ball} is the ball radius.

Tilt to Translation Relationship

Controlled tilt produces ball motion:

$$\ddot{\boldsymbol{p}}_{ball} = g \begin{bmatrix} \tan\theta_y \\ -\tan\theta_x \end{bmatrix} \approx g \begin{bmatrix} \theta_y \\ -\theta_x \end{bmatrix}$$

Principle:

- Tilt forward ($\theta_y > 0$) $\rightarrow$ ball rolls forward
- Tilt right ($\theta_x > 0$) $\rightarrow$ ball rolls right
- Acceleration proportional to tilt angle

Control Strategy

To move in direction $\boldsymbol{v}_{desired}$:

1. Command tilt: $\theta_{cmd} = k_v \boldsymbol{v}_{desired}$
2. Maintain tilt via motor torques
3. Ball accelerates in desired direction

Simplified Nonlinear Model

Complete MIMO underactuated dynamics:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$$

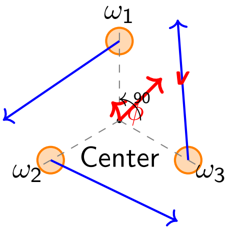
$$\begin{bmatrix} \dot{\theta}_x \\ \dot{\theta}_y \\ \ddot{\theta}_x \\ \ddot{\theta}_y \end{bmatrix} = \begin{bmatrix} \dot{\theta}_x \\ \dot{\theta}_y \\ \omega_n^2 \sin\theta_x - \frac{c_1\tau_1 + c_2\tau_2 + c_3\tau_3}{I} \\ \omega_n^2 \sin\theta_y - \frac{d_1\tau_1 + d_2\tau_2 + d_3\tau_3}{I} \end{bmatrix}$$

where c_i, d_i are kinematic coefficients from $\boldsymbol{J}_{motor}$.

Linearized (small angles):

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}$$

Three-Wheel Kiwi Drive Kinematics



Velocity decomposition:
Global velocity:

$$\boldsymbol{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

Angular velocity: ω_r (rotation)

Goal: Find $[\omega_1, \omega_2, \omega_3]^T$ to achieve $\boldsymbol{v}$ and ω_r

Inverse Kinematics Derivation

Velocity at wheel i perpendicular to mounting:

$$v_{\perp,i} = \boldsymbol{v} \cdot \boldsymbol{n}_i + r_{config}\omega_r$$

where $\boldsymbol{n}_i$ is the unit normal vector for wheel i.

For 120° configuration:

$$\boldsymbol{n}_1 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \quad \boldsymbol{n}_2 = \begin{bmatrix} 1/2 \\ -\sqrt{3}/2 \end{bmatrix}, \quad \boldsymbol{n}_3 = \begin{bmatrix} 1/2 \\ \sqrt{3}/2 \end{bmatrix}$$

Wheel angular velocity:

$$\omega_i = \frac{v_{\perp,i}}{r_{roller}}$$

Matrix Form

Inverse kinematics (body to wheels):

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = \frac{1}{r_{roller}} \begin{bmatrix} -1 & 0 & r_{config} \\ 1/2 & -\sqrt{3}/2 & r_{config} \\ 1/2 & \sqrt{3}/2 & r_{config} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega_r \end{bmatrix}$$

Compact form:

$$\boldsymbol{\omega}_{wheels} = \boldsymbol{J}^{-1} \boldsymbol{v}_{body}$$

Note: For ballbot, ω_r is typically set to 0 (no body spin).
Usage: Desired tilt $\boldsymbol{\theta}_{cmd} \rightarrow \boldsymbol{v}_{body} \rightarrow \boldsymbol{\omega}_{wheels}$

Sensor Fusion: Complementary Filter

MPU-6050 provides:

- Accelerometer: Tilt estimate (noisy, no drift)
- Gyroscope: Angular rate (clean, but drifts)

Complementary filter:

$$\theta_{fused}[k] = \alpha \cdot (\theta_{fused}[k-1] + \omega_{gyro} \cdot dt) + (1-\alpha) \cdot \theta_{accel}$$

where:

- $\alpha \approx 0.98$ (high-pass on gyro, low-pass on accel)
- dt: Sample period
- $\theta_{accel} = \text{atan2}(a_y, a_z)$ for roll

Why Complementary Filter?

- Simple, computationally efficient
- Real-time suitable (Kalman filter is better but more complex)
- Sufficient for ballbot application

PID Control Structure

Independent PID for each axis:

$$\tau_i = K_p e + K_i \int e \, dt + K_d \frac{de}{dt}$$

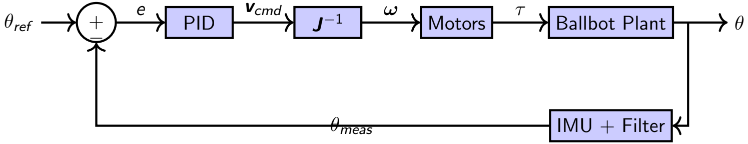
where $e = \theta_{setpoint} - \theta_{measured}$

Discrete implementation:

$$\begin{aligned} P[k] &= K_p \cdot e[k] \\ I[k] &= I[k-1] + K_i \cdot e[k] \cdot dt \\ D[k] &= K_d \cdot \frac{e[k] - e[k-1]}{dt} \\ u[k] &= P[k] + I[k] + D[k] \end{aligned}$$

Anti-windup: Clamp $I[k]$ to prevent integral saturation

Control Block Diagram



Tilt-Based Locomotion Control

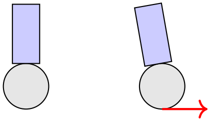
Locomotion via tilt reference:

To move with velocity $\boldsymbol{v}_{desired}$:

$$\theta_{ref} = K_{locomotion} \cdot \boldsymbol{v}_{desired}$$

Example:

- Move forward: $\theta_{y,ref} = +0.1$ rad
- Move right: $\theta_{x,ref} = +0.1$ rad
- Stop: $\theta_{ref} = 0$



Control loop: Outer loop (locomotion) sets $\theta_{ref} \rightarrow$ Inner loop (balance) maintains tilt

Control Loop Structure

Main Control Loop:

1. Read IMU (I2C)
2. Complementary filter
3. Compute PID
4. Inverse kinematics
5. Send step pulses
6. Repeat at fixed rate

Timing Requirements:

- Loop frequency: 100-200 Hz
- Jitter: < 1 ms
- Total latency: < 10 ms

ESP32 Task Separation:

Core 0:

- Sensor reading
- Filter update

Core 1:

- Control computation
- Motor commands
- Communication

Uses FreeRTOS for task management

A4988 Current Limiting

Setting motor current via reference voltage:

$$I_{max} = \frac{V_{ref}}{8 \cdot R_{sense}}$$

For A4988 with $R_{sense} = 0.1\,\Omega$:

$$I_{max} = \frac{V_{ref}}{0.8}$$

Example calculation:

- NEMA17 rated current: $I_{rated} = 1.5$ A
- Set to 70% for margin: $I_{set} = 1.05$ A
- Required V_{ref}: $V_{ref} = 0.8 \times 1.05 = 0.84$ V

Adjustment: Use small screwdriver on potentiometer while measuring V_{ref} with multimeter

Power Consumption and Battery Life

Power budget:

$$\begin{aligned} P_{motors} &= 3 \times V_{supply} \times I_{avg} = 3 \times 11.1 \times 0.8 = 26.6 \text{ W} \\ P_{electronics} &\approx 2 \text{ W} \\ P_{total} &\approx 29 \text{ W} \end{aligned}$$

Battery discharge:
For 3S LiPo, 2200 mAh at 11.1V:

$$E_{battery} = 2.2 \times 11.1 = 24.4 \text{ Wh}$$

Estimated runtime:

$$t_{run} = \frac{24.4}{29} \times 0.8 \approx 40 \text{ minutes}$$

(Factor 0.8 accounts for inefficiency and voltage drop)

PID Gain Selection Effects

Gain	Too Low	Too High	Effect
K_p	Slow response	Oscillation	Proportional
K_i	Steady error	Windup, overshoot	Eliminates bias
K_d	Overshoot	Noise amplification	Damping

Tuning procedure:

1. Start with all gains at zero
2. Increase K_p until small oscillations appear
3. Add K_d to dampen oscillations
4. Add small K_i to eliminate steady-state error
5. Fine-tune iteratively

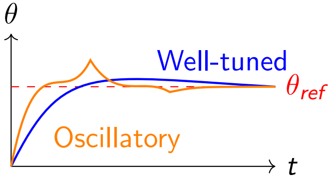
Typical values (normalized):

$$K_p \approx 10\text{-}50, \quad K_i \approx 0.1\text{-}1, \quad K_d \approx 1\text{-}5$$

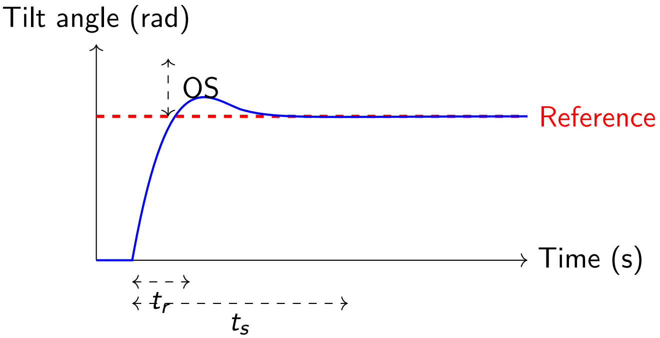
Oscillation Analysis

Types of oscillation:

1. **High-frequency ($>$ 10 Hz):**
   - Cause: Excessive K_d
   - Fix: Reduce K_d, add filter
2. **Medium frequency (2-5 Hz):**
   - Cause: Excessive K_p
   - Fix: Reduce K_p, increase K_d
3. **Low frequency ($<$ 1 Hz):**
   - Cause: Excessive K_i
   - Fix: Reduce K_i, add anti-windup



Step Response Characteristics



Performance metrics:

- Rise time (t_r): Time to reach 90% of reference
- Overshoot (OS): Peak value beyond reference
- Settling time (t_s): Time to stay within $\pm 2\%$ of reference
- Steady-state error: Final error at $t \to \infty$

Current System Limitations

Modeling Simplifications

- Small angle approximation breaks down at $>15^\circ$
- Neglected friction (ball-roller, bearing, aerodynamic)
- Assumed rigid body (ignores flexibility)
- No slip modeling (critical for rapid maneuvers)

Hardware Constraints

- Stepper motor limitations (torque ripple, finite step size)
- IMU noise and bias drift
- Computational latency in ESP32
- Power constraints (battery weight vs. capacity)

Control Limitations

- PID cannot optimize for multiple objectives

Friction and Slip Modeling

Friction torque at ball-roller interface:

$$\tau_{friction} = \mu N r_{contact}$$

where:

- μ: Coefficient of friction (rubber on ball)
- N: Normal force at contact
- $r_{contact}$: Contact radius

Slip condition:
Slip occurs when commanded torque exceeds maximum static friction:

$$|\tau_{cmd}| > \tau_{friction,max}$$

Effects of slip:

- Loss of position tracking
- Reduced control authority
- Potential instability

Mitigation:

Advanced Control: LQR

Linear Quadratic Regulator (LQR):

Minimize cost function:

$$J = \int_0^\infty (\boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} + \boldsymbol{u}^T \boldsymbol{R} \boldsymbol{u})\, dt$$

where:

- $\boldsymbol{Q}$: State cost matrix (penalize deviations)
- $\boldsymbol{R}$: Input cost matrix (penalize control effort)

Optimal control law:

$$\boldsymbol{u}^* = -\boldsymbol{K}\boldsymbol{x}$$

where $\boldsymbol{K}$ is computed by solving the Riccati equation.

Advantages over PID:

- Systematic design (no manual tuning)
- Multi-objective optimization
- Provable stability guarantees

State Estimation: Kalman Filter

Extended Kalman Filter (EKF):

Handles sensor noise and bias:

$$\hat{\boldsymbol{x}}[k|k] = \hat{\boldsymbol{x}}[k|k-1] + \boldsymbol{K}[k](\boldsymbol{z}[k] - \boldsymbol{h}(\hat{\boldsymbol{x}}[k|k-1]))$$

where:

- $\hat{\boldsymbol{x}}$: State estimate
- $\boldsymbol{z}$: Sensor measurements
- $\boldsymbol{K}$: Kalman gain
- $\boldsymbol{h}(\cdot)$: Measurement model

Benefits:

- Optimal fusion of accel and gyro
- Bias estimation and correction
- Better than complementary filter
- Can estimate ball position/velocity

Trade-off: Computational complexity

Field-Oriented Control (FOC) for Motors

Upgrading from steppers to BLDC with FOC:

Advantages:

- Smooth torque output
- Higher efficiency
- Better dynamic response
- Torque control mode
- Higher speed capability

Requirements:

- Current sensing
- Rotor position encoder
- FOC algorithm
- Dedicated motor driver
- Higher computational load

FOC enables:

- Direct torque control (better than velocity control)
- Force/impedance control
- Smoother motion

Future Work and Extensions

Immediate improvements:

- Implement EKF
- Add slip detection
- Vibration damping
- Remote control interface
- Data logging

Medium-term:

- Switch to LQR/LQG
- Add vision system
- Autonomous navigation
- Path planning

Advanced features:

- Machine learning for disturbance rejection
- Adaptive control
- Multi-robot coordination
- Human-robot interaction

Hardware upgrades:

- BLDC with FOC
- Better IMU (BMI088)
- Onboard computer (Jetson Nano)
- LiDAR for obstacle avoidance

Summary

Key Takeaways:

1. Ballbot is an underactuated, inherently unstable system
2. Requires careful mechanical design (120° roller config, high CoM)
3. Mathematical modeling enables systematic control design
4. Sensor fusion (complementary filter) provides reliable state estimate
5. PID control provides balance; tilt commands enable locomotion
6. Hardware selection balances performance, cost, complexity
7. Many opportunities for advanced control and AI integration

Core Principle

Maintain controlled instability to achieve dynamic stability

References and Resources

Key Papers:

- Lauwers et al., "A Dynamically Stable Single-Wheeled Mobile Robot with Inverse Mouse-Ball Drive" (2006)
- Kumagai and Ochiai, "Development of a Robot Balancing on a Ball" (2010)

Software/Hardware:

- ESP32 Arduino Core: `github.com/espressif/arduino-esp32`
- MPU6050 Library: `github.com/jrowberg/i2cdevlib`
- Stepper control: AccelStepper library

Further Learning:

- Slotine & Li, "Applied Nonlinear Control"
- Åström & Murray, "Feedback Systems"
- Lynch & Park, "Modern Robotics"

Thank You!

Questions?

Contact: cipher0xx@gmail.com