CONTROL OF A HEXAPOD ROBOT USING A PID ALGORITHM

Gîlcă Gheorghe, Lecturer Phd., *"Constantin Brâncuşi" University from Târgu Jiu, ROMANIA*

ABSTRACT: The paper presents a method for controlling the motion of a hexapod robot based on a PID (Proportional–Integral–Derivative) algorithm. The proposed system follows predefined trajectories by establishing a two-level control architecture: body-level kinematic control and leg joint control. The advantages of using PID control - such as stability, accuracy, and implementation simplicity - are highlighted.

KEY WORDS: PID control algorithm, desired position, position error, hexapod robot, body-level, joint-level

1. INTRODUCTION

In recent decades, the development of autonomous systems and mobile robots has undergone significant progress, driven by advances in electronics, artificial intelligence, and lightweight high-strength materials. Among the various mechanical configurations used, hexapod robots occupy an important place due to their ability to move stably on uneven terrain, mimicking the locomotion of insects in nature [1].

A hexapod robot is equipped with six articulated legs, each typically having three degrees of freedom (coxa, femur, and tibia). This configuration provides superior static and dynamic stability compared to biped or quadruped robots. Moreover, a hexapod can maintain ground contact even when one or two legs lose traction, making it ideal for applications such as planetary exploration, search and rescue, industrial inspection, or autonomous agriculture [2].

However, the kinematic complexity of a hexapod robot is high—each leg contributes to maintaining balance and generating propulsion, requiring coordinated control of all 18 joints. For this reason, stable trajectory control becomes essential for achieving the desired motion without oscillations or loss of balance.

Controlling the motion of a hexapod robot involves simultaneously managing two levels of complexity:

- **body-level control**, responsible for tracking the desired trajectory in the global workspace, and
- **joint-level control**, responsible for the precise coordination of each leg through inverse kinematics.

Most recent studies rely on advanced control techniques, such as adaptive algorithms, fuzzy models, or neural networks [3]. However, these methods involve a high computational load and complex parameter calibration. In practical applications where hardware resources are limited, a simpler and more robust algorithm is preferable.

The PID (Proportional–Integral–Derivative) algorithm represents a classical yet highly effective solution for controlling systems with moderate nonlinearity. Due to its simple implementation and robustness to parameter variations, PID control is widely used in robotics, including trajectory control, speed control, and joint position regulation [4]. Applying this type of control to a hexapod robot offers a balance between precision, stability, and low computational cost.

In paper [5], the development of an educational system focused on level control in a tank is presented, featuring both manual and automatic adjustment options, allowing the testing of control algorithms such as PID.

In the paper [10] a control system for a dual axis sun tracker is proposed used to automatic or remote position control of the photovoltaic panels.

In paper [6], a detailed method for controlling the movement of a robot participating in a SUMO competition is presented. The article [11] describes the how a robot moves in a labyrinth, and also the possibility of finding the way to the exit of the labyrinth.

The papers [7,12] presents experiments that use voice control applied to animatronic structures as well as to a robotic structure.

2. MODELING OF THE HEXAPOD ROBOT

This chapter explains the mechanical structure, the kinematic model (both forward and inverse), and the locomotion principles of a hexapod robot.

2.1 Mechanical Structure of the Hexapod Robot

A hexapod robot is a platform equipped with six independent legs, each having three degrees of freedom (DOF):

- **coxa** – the proximal joint that rotates the leg in the horizontal plane;
- **femur** – the middle segment that controls the lifting or lowering of the leg;
- **tibia** – the distal segment that extends or contracts the step length [1].

Thus, the robot has a total of **18 degrees of freedom**.

The central body (referred to as the *body* or *chassis*) serves as a platform for mounting sensors (IMU, cameras, LiDAR) and the power supply.

2.1.1 Main Structural Components

- Base structure: made of aluminum or composite materials to ensure low weight and high rigidity.
- Actuators: digital servo motors (e.g., Dynamixel MX-64) that provide angular precision and torque control.
- Sensors: joint encoders, IMU (accelerometer + gyroscope) for body orientation, and optionally contact sensors for ground detection.
- Controller: a microcontroller (STM32, Arduino Mega) or a mini-computer (Raspberry Pi, Jetson Nano) equipped with CAN or serial interfaces for leg synchronization.

2.1.2 Leg positioning

The legs are arranged symmetrically, three on each side of the body:

- **front legs (1,2):** responsible for directional stabilization;
- **middle legs (3,4):** responsible for maintaining the center of mass;
- **rear legs (5,6):** responsible for propulsion.

This configuration provides the robot with a stable support polygon, even during locomotion [8].

2.2 Kinematic model

The kinematic model of a hexapod robot describes the relationship between the joint angles and the spatial position of the leg's end-effector.

2.2.1 Forward kinematics of the leg

For each leg, the following segments are defined:

- L_1: length of the coxa;
- L_2: length of the femur;
- L_3: length of the tibia.

The position of the leg's end-effector in the local coordinate system is given by:

$$\begin{aligned} x_f &= L_1\cos q_1 + L_2\cos(q_1+q_2) + L_3\cos(q_1+q_2+q_3), \\ y_f &= L_1\sin q_1 + L_2\sin(q_1+q_2) + L_3\sin(q_1+q_2+q_3), \\ z_f &= 0, \end{aligned} \quad (1)$$

where q_1, q_2, and q_3 are the joint [9].

2.2.2 Inverse kinematics of the leg

The inverse kinematics problem consists in determining the joint angles (q_1, q_2, q_3) for a desired end-effector position (x_f, y_f, z_f).

The typical steps are:

1) **Determine angle q_1** from the projection onto the horizontal plane:

$$q_1 = \arctan\ 2(y_f, x_f) \quad (2)$$

2) Apply the law of cosines to determine q_2 and q_3:

$$D = \frac{x_f^2 + y_f^2 - L_2^2 - L_3^2}{2L_2L_3} \quad (3)$$

$$q_3 = \arctan\ 2(-\sqrt{1-D^2}, D) \quad (4)$$

$$q_2 = \arctan\ 2(y_f, x_f) - \arctan\ 2(L_3\sin\ q_3, L_2 + L_3\cos\ q_3) \quad (5)$$

These relations are used to compute the command values for each joint at every control cycle [4].

2.3 Body Kinematics

The central body of the hexapod is considered a rigid planar platform characterized by the coordinates: (x, y, θ),
where x and y are the positions in the plane, and θ is the orientation relative to the global axis.

The motion of the body is described by:

$$\dot{x} = v_x\cos\ \theta - v_y\sin\ \theta, \dot{y} = v_x\sin\ \theta + v_y\cos\ \theta, \dot{\theta} = \omega, \quad (6)$$

where (v_x, v_y) represent the translational velocities, and ω the angular velocity [8].
Pentru o deplasare stabilă, traiectoria corpului trebuie să fie compatibilă cu traiectoriile picioarelor - adică proiecția centrului de greutate (CoG) să rămână în interiorul poligonului de sprijin [2].

3. PID CONTROL ALGORITHM

Controlling a hexapod robot requires the synchronization of two distinct levels:

1. **Body Control** – establishing the global motion of the robot: position (x,y) and orientation θ .
2. **Joint Control** – accurately positioning each leg according to the desired trajectories.

The PID (Proportional–Integral–Derivative) algorithm is one of the most widely used and effective control methods for moderately nonlinear systems, such as legged robots [4]. This chapter presents the mathematical model of the PID controller, tuning techniques, and its integration into the hexapod control system.

3.1 Mathematical model of the PID controller

The PID controller generates the control signal u(t) based on the error between the desired value r(t) and the measured value y(t):

$$e(t) = r(t) - y(t) \quad (7)$$

The control signal is:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de(t)}{dt} \quad (8)$$

where:

- K_p = proportional gain – reacts instantly to the error;
- K_i = integral gain – eliminates steady-state error;
- K_d = derivative gain – attenuates oscillations and increases stability.

Discrete version (used in digital controllers)
At a control frequency of 50–200 Hz:

$$u[k] = K_p e[k] + K_i T \sum_{i=0}^{k} e[i] + K_d \frac{e[k] - e[k-1]}{T} \quad (9)$$

where T is sampling period.

3.2 Application of the PID Controller to the Hexapod

The PID control is integrated at two levels:

3.2.1 Body-Level PID Controller

Objective: the robot must follow a global trajectory defined by:

$$\mathbf{p_d} = (x_d, y_d, \theta_d) \quad (10)$$

The error in global coordinates:

$$e_x = x_d - x$$
$$e_y = y_d - y \quad (11)$$
$$e_\theta = \theta_d - \theta$$

To avoid singularities, the errors are transformed into the body frame:

$$e_x^b = \cos\theta \cdot e_x + \sin\theta \cdot e_y$$
$$e_y^b = -\sin\theta \cdot e_x + \cos\theta \cdot e_y \quad (12)$$

PID applied to each error:

$$v_x = K_{px} e_x^b + K_{ix} \int e_x^b dt + K_{dx} \frac{de_x^b}{dt}$$
$$v_y = K_{py} e_y^b + K_{iy} \int e_y^b dt + K_{dy} \frac{de_y^b}{dt} \quad (13)$$
$$\omega = K_{p\theta} e_\theta + K_{i\theta} \int e_\theta dt + K_{d\theta} \frac{de_\theta}{dt}$$

The result is a set of reference velocities ($\upsilon_x, \upsilon_y, \omega$) that are sent to the gait planner.

3.2.2 Joint-Level PID

For each leg:

$$\mathbf{q_d} = (q_{1d}, q_{2d}, q_{3d}) \quad (14)$$

where the values are generated by the inverse kinematics.

The error for each joint:

$$e_i = q_{id} - q_i \quad (15)$$

The PID controller is applied:

$$u_i = K_{p_i} e_i + K_{i_i} \int e_i dt + K_{d_i} \frac{de_i}{dt} \quad (16)$$

This generates the torque or position command for the joint servomotors.

3.3 PID parameter tuning

Proper tuning of the PID parameters is essential for stable locomotion. There are three main methods:

3.3.1 Manual (Heuristic) Tuning

1. Increase $\mathbf{K_p}$ until oscillations appear.
2. Increase $\mathbf{K_d}$ to reduce the oscillations.
3. Increase $\mathbf{K_i}$ only as much as necessary to eliminate the steady-state error.

Advantage: simple and fast. **Disadvantage:** depends on user experience.

3.3.2 Ziegler–Nichols (ZN) Method

1. Set $\mathbf{K_i = 0}$ and $\mathbf{K_d = 0}$.
2. Gradually increase $\mathbf{K_p}$ until the system reaches sustained oscillations → obtain:
   - $\mathbf{K_u}$ – ultimate (critical) gain,
   - $\mathbf{T_u}$ – oscillation period.

Table ZN:

Tip control	K_p	K_i	K_d
P	$0.50K_u$	–	–
PI	$0.45\ K_u$	$1.2K\_p/T_u$	–
PID	$0.60\ K_u$	$2K_p/\ T_u$	$K_pT_u/8$

The ZN method is a standard approach in the robotics control literature.

3.3.3 Adaptive PID tuning for a hexapod robot

For uneven terrain, the PID gains can be adjusted in real time:

$$K_p(t) = K_{p0} + \alpha \mid e(t) \mid$$
$$K_d(t) = K_{d0} + \beta \mid \dot{e}(t) \mid \quad (17)$$

This method enables stable control even when the load or terrain varies [3].

4. SOFTWARE ARCHITECTURE

This section presents in detail the software architecture used for implementing PID control of the hexapod robot, as well as the results obtained in the simulation environment. The main objective is to validate the controller's performance under controlled conditions, prior to implementation on real hardware.

```
//============================== Gait ================================

class TripodGait {
public:
  TripodGait(double cycle=0.6, double duty=0.6, double clearance=0.025, double stepLen=0.06)
  : T_(cycle), duty_(duty), clr_(clearance), step_(stepLen) {
    groupA_ = {1,4,5}; groupB_ = {2,3,6};
  }
  // phase in [0,1)
  double phase(int legId, double t) const {
    double tau = fmod(t, T_) / T_;
    if (groupB_.count(legId)) tau = fmod(tau + 0.5, 1.0);
    return tau;
  }
  // target foot point in leg local frame (simple nominal pattern)
  std::array<double,3> footTarget(int legId, double t, double vx_body) const {
    (void)vx_body; // keep simple; could scale step_ by vx_body
    double tau = phase(legId, t);
    const double stance = duty_;
    const double x_back = -step_/2.0, x_front = +step_/2.0;
    const double y_off = 0.0, z0 = 0.0;

    double x,y,z;
    if (tau < stance) {
      double s = tau / stance;
      x = x_front + (x_back - x_front) * s;
      y = y_off; z = z0;
    } else {
      double s = (tau - stance) / (1.0 - stance);
      // smoothstep quintic: 10s^3 - 15s^4 + 6s^5
      double s2=s*s, s3=s2*s, s4=s3*s, s5=s4*s;
      double blend = 10*s3 - 15*s4 + 6*s5;
      x = x_back + (x_front - x_back) * blend;
      y = y_off;
      z = z0 + clr_ * sin(PI * s); // arc over ground
    }
    return {x,y,z};
  }
private:
  double T_, duty_, clr_, step_;
  std::set<int> groupA_, groupB_;
};

//============================== Body Control ============================

struct Pose2D { double x{0}, y{0}, th{0}; };
struct BodyCmd { double vx{0}, vy{0}, wz{0}; };

class BodyPID {
public:
  BodyPID(double dt) {
    PIDGains gx{1.2, 0.1, 0.3, 20.0, -0.25, 0.25, -0.5, 0.5};
    PIDGains gy{1.2, 0.1, 0.3, 20.0, -0.25, 0.25, -0.5, 0.5};
    PIDGains gt{2.0, 0.1, 0.2, 20.0, -1.2, 1.2, -0.5, 0.5};
    px_ = PID(gx, dt); py_ = PID(gy, dt); pth_ = PID(gt, dt);
  }
  BodyCmd compute(const Pose2D& cur, const Pose2D& ref) {
    double dx = ref.x - cur.x;
    double dy = ref.y - cur.y;
    double dth = wrap_pi(ref.th - cur.th);
    double c = cos(cur.th), s = sin(cur.th);
    double exb =  c*dx + s*dy;
    double eyb = -s*dx + c*dy;
    return { px_.update(exb), py_.update(eyb), pth_.update(dth) };
  }
private:
  PID px_, py_, pth_;
};
```

```
//============================ Joint Control ============================
class LegJointPID {
public:
  LegJointPID(double dt) {
    // initial gains; tune for your platform
    pids_[0] = PID(PIDGains{6.0, 30.0, 0.03, 25.0, -2.0, 2.0, -0.2, 0.2}, dt); // coxa
    pids_[1] = PID(PIDGains{8.0, 40.0, 0.05, 25.0, -2.5, 2.5, -0.2, 0.2}, dt); // femur
    pids_[2] = PID(PIDGains{7.0, 35.0, 0.04, 25.0, -2.5, 2.5, -0.2, 0.2}, dt); // tibia
  }
  std::array<double,3> torque(const std::array<double,3>& q,
                              const std::array<double,3>& qd) {
    std::array<double,3> u{};
    for (int i=0;i<3;i++) {
      double e = qd[i] - q[i];
      u[i] = pids_[i].update(e);
    }
    return u;
  }
private:
  std::array<PID,3> pids_;
};
//============================ Hexapod Controller ======================
class HexapodController {
public:
  explicit HexapodController(double dt=0.005) : dt_(dt), body_(dt), gait_() {
    geom_ = {0.05, 0.10, 0.12};
    limits_ = {
      {deg2rad(-60), deg2rad( 60)},
      {deg2rad(-10), deg2rad( 80)},
      {deg2rad(-130), deg2rad(-5)}
    };
    for (int i=1;i<=6;i++) {
      ik_[i] = std::make_unique<LegIK>(geom_, limits_);
      joints_[i] = std::make_unique<LegJointPID>(dt_);
    }
    // nominal mounts (if you later transform targets from body to leg frames)
    mountXY_ = {
      {1, {+0.10, +0.09}},
      {2, {+0.10, -0.09}},
      {3, { 0.00, +0.11}},
      {4, { 0.00, -0.11}},
      {5, {-0.10, +0.09}},
      {6, {-0.10, -0.09}},
    };
  }

  // ---- Provide pose from sensors or simulator ----
  void setPoseForSim(const Pose2D& p){ pose_ = p; }
  Pose2D readPose() const { return pose_; }

  // ---- Main control step ----
  void step(double t, const Pose2D& ref) {
    BodyCmd cmd = body_.compute(pose_, ref);

    // 1) (Optional) adjust gait using cmd (vx,vy,wz)
    // 2) For each leg: generate foot target, solve IK, compute torques, apply
    for (int leg=1; leg<=6; ++leg) {
      auto tgt = gait_.footTarget(leg, t, cmd.vx);
      auto qd  = ik_[leg]->solve(tgt[0], tgt[1], tgt[2]);
      LegState st = readJointState(leg);          // TODO: replace
      auto tau = joints_[leg]->torque(st.q, qd);
      applyJointTorque(leg, tau);                 // TODO: replace
    }

    // 3) Tiny body kinematics "sim" so the demo runs standalone
    integrateBody(cmd);
  }

private:
  // -------- Hardware/simulator hooks (replace for your system) --------
  void applyJointTorque(int leg, const std::array<double,3>& u) {
    (void)leg; (void)u;
    // TODO: send torques to drivers or physics engine
  }
  LegState readJointState(int leg) const {
    (void)leg;
    // TODO: read encoders; here we return zeros as a placeholder
    return {};
  }
  // ---------------------------------------------------------------------

  void integrateBody(const BodyCmd& cmd) {
    // Integrate simple 2D body kinematics (world frame)
    double c = cos(pose_.th), s = sin(pose_.th);
    double dx = ( cmd.vx*c - cmd.vy*s ) * dt_;
    double dy = ( cmd.vx*s + cmd.vy*c ) * dt_;
    double dth= cmd.wz * dt_;
    pose_.x += dx; pose_.y += dy; pose_.th = wrap_pi(pose_.th + dth);
  }

  static double deg2rad(double d){ return d*PI/180.0; }

  double dt_;
  Pose2D pose_{};
  BodyPID body_;
  TripodGait gait_;
  LegGeom geom_;
  LegLimits limits_;
  std::map<int, std::unique_ptr<LegIK>> ik_;
  std::map<int, std::unique_ptr<LegJointPID>> joints_;
  std::map<int, std::array<double,2>> mountXY_;
};

//============================ Reference Trajectory ====================

static Pose2D refTrajectory(double t) {
  // 0..8s: +X (0.2 m/s), 8..16s: +Y, 16..20s: rotate to +90 deg
  if (t < 8.0)      return {0.2*t, 0.0, 0.0};
  else if (t < 16.0) return {1.6, 0.2*(t-8.0), 0.0};
  else {
    double th = std::min(90.0, (t-16.0)*22.5);
    return {1.6, 1.6, th * PI/180.0};
  }
}

//============================== Main ==============================

int main() {
  const double dt = 0.005; // 200 Hz
  HexapodController hex(dt);
  hex.setPoseForSim({0.0, 0.0, 0.0});

  auto t0 = chrono::steady_clock::now();
  double simT = 20.0;
  while (true) {
    auto now = chrono::steady_clock::now();
    double t = chrono::duration<double>(now - t0).count();
    if (t > simT) break;

    Pose2D ref = refTrajectory(t);
    hex.step(t, ref);

    // simple pacing (not hard RT)
    std::this_thread::sleep_for(std::chrono::milliseconds(4)); // ~250 Hz loop pacing
  }

  Pose2D p = hex.readPose();
  std::cout << "Final pose: x=" << p.x << " m, y=" << p.y
        << " m, th=" << (p.th*180.0/PI) << " deg\n";
  return 0;
}
```

5. CONCLUSIONS

Implementing PID control for a hexapod robot represents a simple, robust, and efficient solution for educational, academic, and proto-industrial applications. By extending the system with adaptive or predictive methods, performance comparable to that of advanced robots used in rough-terrain exploration can be achieved.

Compared to P and PI control, PID control is clearly superior for hexapod robots because it:

- reduces tracking error,
- improves stability,
- ensures robustness against disturbances,
- maintains reasonable energy consumption.

REFERENCES

[1] Corke, P. (2017). Robotics, Vision and Control: Fundamental Algorithms in MATLAB. Springer.

[2] Liu, Y., & Wang, J. (2018). "Gait planning and trajectory control of a hexapod robot." International Journal of Advanced Robotic Systems.

[3] Ryu, J., & Kim, D. (2020). "A study on adaptive PID control for multi-legged robots." Mechanisms and Machine Theory, Elsevier.

[4] Guo, Y., Li, H., & Sun, T. (2021). "PID control for hexapod robots using body-level trajectory tracking." IEEE Transactions on Robotics.

[5] Florin Grofu. "Educational system for studying level control in a tank", Fiability & Durability / Fiabilitate si Durabilitate. mai 2025, Vol. 35 Issue 1, pp 316-319.

[6] Ilie Borcosi, The control of a SUMO robot, Analele UCB, Seria Inginerie, nr.4/2017, ISSN 1842-4856, pag. 140-144.

[7] Ionescu M., Using voice commands to obtain expressive states for animatronic structures, Fiabilitate si Durabilitate - Fiability & Durability No 1/ 2022, Editura "Academica Brâncuşi" , Târgu Jiu, ISSN 1844 – 640X, Engineering Series, pp 127-134.

[8] Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2010). Robotics: Modelling, Planning and Control. Springer.

[9] McGhee, R. B., & Frank, A. A. (1968). "On the stability properties of quadruped and hexapod walking machines." Mathematical Biosciences.

[10] Grofu Florin, "Control System For Photovoltaic Panels Tracker" Revista de Fiabilitate și Durabilitate Nr 1(21)/2018 Editura "Academica Brâncuşi" , Târgu Jiu, ISSN 1844 – 640X Pg. 333-338

[11] Ilie Borcoși, THE MOVING OF A ROBOT IN THE LABYRINTH, Annals of the „Constantin Brancusi" University of Targu Jiu, Engineering Series, No. 4/2016, pp. 120-122, ISSN 1842-4856.

[12] Ionescu, M., I Borcosi, N G Bizdoaca, Voice reactive biomimetic structure, The 8th International Conference on Advanced Concepts in Mechanical Engineering - ACME 2018, organized by Mechanical Engineering Faculty, in the "Gheorghe Asachi" Technical University of Iasi, Romania, June 07 - 08, 2018, pp.457-466, WOS:000467443600081. https://www.proceedings.com/content/042/042496webtoc.pdf