Closed-Loop Control of a Differential-Drive Robot Using Jacobian-Based Cartesian Feedback

Harsha Abeykoon, PhD
Associate Professor
Embry-Riddle Aeronautical University

1 Introduction

In traditional mobile robotics courses, differential-drive robots are often controlled using low-level wheel velocities or simple linear and angular velocity commands. However, it is also possible to control a mobile robot in the Cartesian (x, y) space by drawing an analogy to robot manipulators. This document presents a closed-loop control approach for a differential-drive robot using Jacobian-based Cartesian feedback. The method illustrates how to treat the mobile robot like a two-degree-of-freedom planar manipulator, where the "end-effector" is the robot's chassis position.

This approach is inspired by my prior research that applied Jacobian-based control to mobile robots. In particular, the work of Dayarathna *et al.* [1] we demonstrated precise trajectory control for a differential-drive robot (in the context of pushing an object) by estimating contact forces and controlling the robot's path. Building on that concept, here I simplify and adapt the idea for teaching purposes, focusing on pure odometric (position-based) feedback control.

2 Coordinate Frames and Kinematics

Figure 1 illustrates the coordinate frames and geometry of the differential-drive robot. The robot has two wheels separated by a baseline (wheel track) of W (meters). We define the robot's pose as (x, y, θ) in a global frame: x and y give the position of the midpoint between the wheels, and θ is the robot's heading angle measured from the global X-axis. The robot's body frame (x, y) is attached to the chassis, with x pointing forward and y pointing sideways. At $\theta = 0$, the robot's x axis aligns with the global X-direction.

Each wheel can be driven independently. Let v_L and v_R be the linear velocities of the left and right wheels, respectively (positive if rolling forward). These relate to the robot's translational and rotational velocity. The instantaneous kinematics of the differential-drive robot (assuming no wheel slip) are given by:

$$\dot{x} \;=\; \frac{v_R + v_L}{2}\cos\theta, \qquad \dot{y} \;=\; \frac{v_R + v_L}{2}\sin\theta, \qquad \dot{\theta} \;=\; \frac{v_R - v_L}{W}\,. \tag{1}$$

Equation (1) describes how the wheel motions produce a velocity $(\dot{x}, \dot{y})$ in the global frame and an angular velocity $\dot{\theta}$. In matrix form, these equations can be expressed using a Jacobian matrix J. Defining the state velocity vector $\dot{\mathbf{q}} = [\,\dot{x},\ \dot{y},\ \dot{\theta}\,]^T$ and the wheel velocity vector $\mathbf{v}_{wheel} = [\,v_L,\ v_R\,]^T$, we have:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{1}{2}\cos\theta & \frac{1}{2}\cos\theta \\ \frac{1}{2}\sin\theta & \frac{1}{2}\sin\theta \\ -\frac{1}{W} & \frac{1}{W} \end{pmatrix}}_{J(\theta)} \begin{pmatrix} v_L \\ v_R \end{pmatrix}. \tag{2}$$

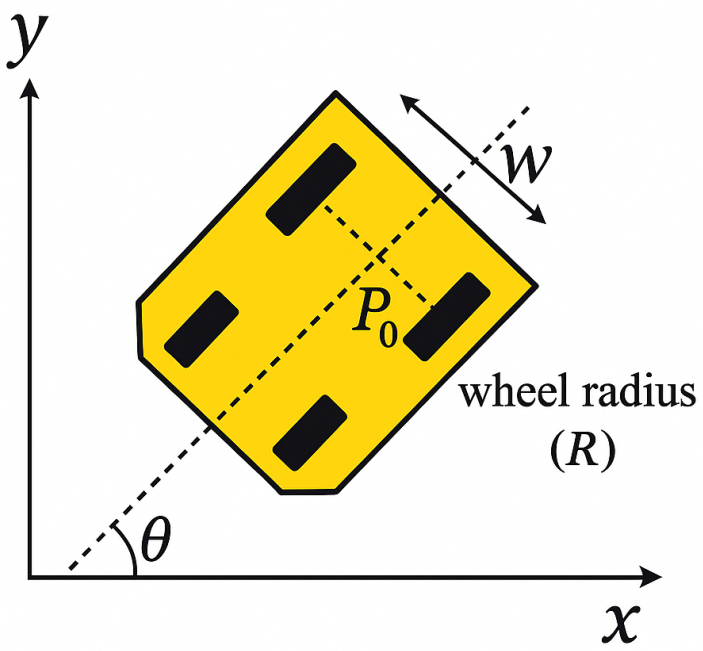

Figure 1: Coordinate frame of a differential-drive mobile robot in the world frame. P_0 denotes the robot's center position, ϕ is the heading angle, w is the wheel separation, and R is the wheel radius.

In the Jacobian $J(\theta)$ above, the first and second rows indicate how wheel motions contribute to $\dot{x}$ and $\dot{y}$ (these depend on the robot's orientation θ via $\cos\theta$ and $\sin\theta$), and the third row describes how differential wheel speeds contribute to $\dot{\theta}$. Note that J is a 3×2 matrix because the robot has 3 degrees of freedom (planar position and orientation) but only 2 wheel inputs. The rank of J is 2 (for any θ where $\cos\theta$ and $\sin\theta$ are not both zero), indicating the system is locally controllable in 2 independent directions. Intuitively, a differential-drive robot cannot move directly sideways; at any instant its velocity is constrained to lie along its current heading.

3 Jacobian Inverse Kinematics (Pseudoinverse)

To control the robot's motion in the (x, y) plane, we need to invert the kinematic relationship (2) to find the required wheel speeds for a desired Cartesian velocity. Since J is not square, we use the Moore–Penrose pseudoinverse of J (denoted J^+) to map Cartesian velocities to wheel velocities. Given a desired state-velocity vector $\dot{\mathbf{q}}_d = [\,\dot{x}_d,\ \dot{y}_d,\ \dot{\theta}_d\,]^T$, the least-squares-optimal wheel speeds can be obtained by:

$$\begin{pmatrix} v_L \\ v_R \end{pmatrix}_d \;=\; J^+\,\dot{\mathbf{q}}_d\,. \tag{3}$$

For the differential-drive $J(\theta)$ (which is full column rank), the pseudoinverse can be computed as

$$J^+ \;=\; (J^T J)^{-1} J^T\,,$$

yielding a 2×3 matrix. Expanding this for $J(\theta)$ in (2) gives explicit formulas for v_L and v_R in terms of $\dot{x}_d$, $\dot{y}_d$, and $\dot{\theta}_d$. In the special case where we only specify a desired translational velocity (i.e. we have targets

for $\dot{x}$ and $\dot{y}$ but set $\dot{\theta}_d = 0$), the pseudoinverse will automatically choose wheel speeds that minimize error in the unattainable direction (lateral motion) by appropriately turning the robot. The pseudoinverse thus generalizes the inverse-kinematics solution for this nonholonomic platform.

4 Closed-Loop Kinematics with PD Feedback

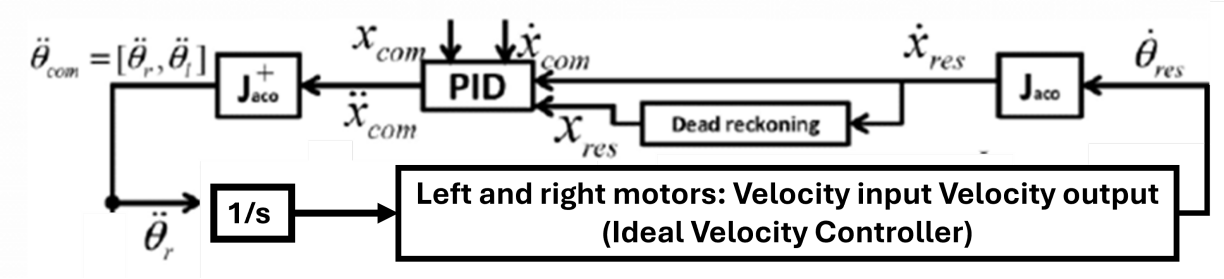

Figure 2: Overall closed-loop control architecture used for the differential-drive robot. The controller operates in Cartesian (x, y) space, generates desired accelerations via PD, integrates to obtain commanded velocities, maps these through the Jacobian pseudoinverse to wheel commands, and reconstructs the robot motion through forward kinematics and dead-reckoning.

This section describes the complete closed-loop control and kinematic reconstruction used in the simulation. The controller operates in the (x, y, θ) space, generates desired accelerations, integrates these to obtain desired body velocities, maps them to wheel commands, and reconstructs the robot motion through dead-reckoning. The structure matches exactly the implemented Python code.

4.1 PD Feedback in (x, y, θ) Space

Tracking errors are defined as

$$e_x = x_{\text{ref}} - x, \qquad e_y = y_{\text{ref}} - y, \qquad e_\theta = \theta_{\text{ref}} - \theta.$$

Let $\dot{x}_{\text{ref}}, \dot{y}_{\text{ref}}, \dot{\theta}_{\text{ref}}$ denote reference velocities. The PD controller computes desired accelerations as

$$\ddot{x}_d = k_p\, e_x \;+\; k_v\, (\dot{x}_{\text{ref}} - \dot{x}),$$

$$\ddot{y}_d = k_p\, e_y \;+\; k_v\, (\dot{y}_{\text{ref}} - \dot{y}),$$

$$\ddot{\theta}_d = k_p\, e_\theta \;+\; k_v\, (\dot{\theta}_{\text{ref}} - \dot{\theta}).$$

4.2 Integration to Obtain Desired Velocities

Desired accelerations are integrated to obtain desired velocities:

$$\dot{x}_d(t + \Delta t) = \dot{x}_d(t) + \ddot{x}_d\, \Delta t,$$

$$\dot{y}_d(t + \Delta t) = \dot{y}_d(t) + \ddot{y}_d\, \Delta t,$$

$$\dot{\theta}_d(t + \Delta t) = \dot{\theta}_d(t) + \ddot{\theta}_d\, \Delta t.$$

These $\dot{x}_d, \dot{y}_d, \dot{\theta}_d$ act as the commanded body velocities.

4.3 Inverse Kinematics via the Jacobian Pseudoinverse

The differential-drive inverse kinematics used in the code is

$$\begin{bmatrix}\omega_{L,d}\\ \omega_{R,d}\end{bmatrix} = J^{+}\begin{bmatrix}\dot{x}_d\\ \dot{y}_d\\ \dot{\theta}_d\end{bmatrix},$$

where

$$J^{+} = \begin{bmatrix}\frac{\cos\theta}{R} & \frac{\sin\theta}{R} & \frac{w}{2R}\\ \frac{\cos\theta}{R} & \frac{\sin\theta}{R} & -\frac{w}{2R}\end{bmatrix}.$$

4.4 Ideal Motors: Commanded = Measured Wheel Velocities

The simulation assumes perfect low-level motor control:

$$\omega_L = \omega_{L,d}, \qquad \omega_R = \omega_{R,d}.$$

Thus commanded wheel velocities are treated as the actual measured values.

4.5 Forward Kinematics and Jacobian

The forward kinematic mapping is

$$\dot{\mathbf{q}} = J\,\boldsymbol{\omega}, \qquad \mathbf{q} = [x\ y\ \theta]^T, \qquad \boldsymbol{\omega} = [\omega_L\ \ \omega_R]^T,$$

where the Jacobian is

$$J = \begin{bmatrix}\frac{R}{2}\cos\theta & \frac{R}{2}\cos\theta\\ \frac{R}{2}\sin\theta & \frac{R}{2}\sin\theta\\ \frac{R}{w} & -\frac{R}{w}\end{bmatrix}.$$

Expanding the Jacobian multiplication gives the actual robot velocities:

$$\dot{x} = \frac{R}{2}(\omega_L + \omega_R)\cos\theta,$$

$$\dot{y} = \frac{R}{2}(\omega_L + \omega_R)\sin\theta,$$

$$\dot{\theta} = \frac{R}{w}(\omega_L - \omega_R).$$

4.6 Dead-Reckoning Integration of Actual Motion

The robot pose is updated using numerical integration:

$$x(t+\Delta t) = x(t) + \dot{x}(t)\Delta t,$$

$$y(t+\Delta t) = y(t) + \dot{y}(t)\Delta t,$$

$$\theta(t+\Delta t) = \theta(t) + \dot{\theta}(t)\Delta t.$$

These integrated states provide the dead-reckoned position and orientation, which are fed back to compute the next tracking errors (e_x, e_y, e_θ), thereby closing the loop.

4.7 Closed-Loop Structure

The full loop executed in the simulation is:

1. Compute tracking errors (e_x, e_y, e_θ).
2. PD feedback produces desired accelerations.
3. Integrate accelerations to obtain desired velocities.
4. Apply inverse kinematics J^+ to compute wheel commands.
5. Ideal motors ensure wheel response equals wheel command.
6. Forward kinematics compute actual velocities.
7. Dead-reckoning integrates the actual robot pose.
8. Updated pose closes the feedback loop.

4.8 Time Vector and State Initialization

The simulation is executed over a discrete time horizon. The time vector is defined as

$$t = \{0,\ T_s,\ 2T_s,\ 3T_s, \ldots, T\},$$

where T is the total simulation duration and T_s is the sampling interval. In code this is generated by

```
t = np.arange(0, T, Ts);
N = len(t);
```

Thus, N denotes the total number of simulation steps.

4.9 Commanded Trajectory $(x_{\mathbf{com}}, y_{\mathbf{com}}, \phi_{\mathbf{com}})$

The reference (commanded) path is a sinusoidal trajectory defined as

$$x_{\text{com}}(t) = vt, \qquad y_{\text{com}}(t) = A\sin(kx_{\text{com}}(t)).$$

Its corresponding commanded velocities are

$$\dot{x}_c(t) = v, \qquad \dot{y}_c(t) = Akv\cos(kx_{\text{com}}(t)).$$

The commanded heading angle is computed using

$$\phi_{\text{com}}(t) = \arctan 2(\dot{y}_c(t),\, \dot{x}_c(t))\,,$$

which gives the instantaneous orientation required to follow the path.

The commanded angular velocity is obtained numerically:

$$\dot{\phi}_c(t_i) = \frac{\phi_{\text{com}}(t_i) - \phi_{\text{com}}(t_{i-1})}{T_s}, \qquad i \geq 1.$$

The corresponding code fragment is:

```
x_com   = v * t
y_com   = A * np.sin(k * x_com)

x_dot_c = v * np.ones(N)
y_dot_c = A * k * v * np.cos(k * x_com)

phi_com = np.arctan2(y_dot_c, x_dot_c)
phi_dot_c[1:] = (phi_com[1:] - phi_com[:-1]) / Ts
```

4.10 Response Arrays for Actual Robot Motion

The arrays storing the reconstructed (actual) robot trajectory are initialized as

$$x_{\text{res}}(0) = 0, \qquad y_{\text{res}}(0) = 0, \qquad \phi_{\text{res}}(0) = 0.$$

Velocity arrays are similarly initialized:

$$\dot{x}_{\text{res}}(0) = 0, \qquad \dot{y}_{\text{res}}(0) = 0, \qquad \dot{\phi}_{\text{res}}(0) = 0.$$

In code:

```
x_res   = np.zeros(N)
y_res   = np.zeros(N)
phi_res = np.zeros(N)

dx_res   = np.zeros(N)
dy_res   = np.zeros(N)
dphi_res = np.zeros(N)
```

These variables are filled through forward kinematics and dead-reckoning at each time step.

4.11 Wheel State Initialization

The wheel angles and angular velocities start at zero,

$$\theta_L(0) = 0, \qquad \theta_R(0) = 0,$$

$$\dot{\theta}_L(0) = 0, \qquad \dot{\theta}_R(0) = 0.$$

Code initialization:

```
theta_r = 0.0
theta_l = 0.0
theta_dot_r = 0.0
theta_dot_l = 0.0
```

This represents a robot starting from rest with zero wheel rotation and zero body velocity.

5 Conclusion

Using a Jacobian-based Cartesian feedback controller offers a unique perspective in mobile robot control, linking concepts from manipulator kinematics to wheeled robots. By deriving the Jacobian matrix and its pseudoinverse, students can see how desired motions in the plane translate into wheel commands. The attached simulation code (see accompanying GitHub repository) demonstrates this control loop in action, allowing students to experiment with gain values and observe the robot's trajectory. This exercise reinforces understanding of differential-drive kinematics, odometry, and feedback control in a practical and engaging manner.

References

[1] H. A. N. D. Dayarathna, L. L. G. Prabuddha, K. L. D. N. J. Ariyawansha, M. K. C. D. Chinthaka, A. M. H. S. Abeykoon, and M. B. Pillai, *Sensorless contact position estimation of a mobile robot in pushing motion*, in *Proc. 2013 Int. Conf. on Circuits, Power and Computing Technologies (ICCPCT)*, pp. 344–349, IEEE, 2013.