Robotics Practical Example: From Theory to Simulation

1. Introduction to Robotic Systems

This module presents a structured approach to understanding and implementing robotic system simulations. We'll progress from theoretical foundations to practical simulation implementation, covering mobile robots.

Learning Objectives:

- Understand the fundamental differences between kinematic and dynamic models
- Derive and implement equations of motion for robotic systems
- Translate theoretical models into simulation code
- Analyze simulated robot behaviour and validate against theoretical predictions

2. Mobile Robot Systems: Theory to Simulation

2.1 Theoretical Foundations

2.1.1 Kinematic Model of Differential Drive Robot

Kinematics deals with the motion of bodies without considering the forces that cause them. For mobile robots, especially wheeled ones, kinematic modeling involves understanding how wheel motions translate to robot movement.

A differential drive robot uses two independently controlled wheels to achieve motion. The kinematic model describes the geometrical relationships without considering forces and inertia.

State Variables:

- Position: (x, y) in world frame
- Orientation: θ (heading angle)
- Wheel angular velocities: ω_L (left wheel), ω_R (right wheel)

Forward Kinematics Equations: The robot's linear velocity v and angular velocity ω can be computed from wheel velocities:

$$v = \frac{v_L + v_R}{2} \tag{1}$$

$$\omega = \frac{v_R - v_L}{L} \tag{2}$$

With the linear velocities of the wheels are:

$$v_L = r \cdot \omega_L \tag{3.1}$$

$$v_R = r \cdot \omega_R \tag{3.2}$$

where:

- r is the wheel radius
- L is the distance between wheels (wheel base)

Motion Equations: The position and orientation change according to:

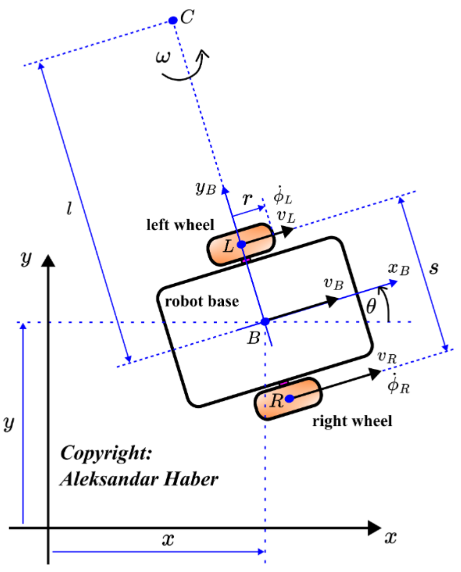
$$\dot{x} = v \cdot \cos\theta \quad (4.1)$$
$$\dot{y} = v \cdot \sin\theta \quad (4.2)$$
$$\dot{\theta} = \omega \quad (4.3)$$

These equations describe how the robot's position and orientation change based on wheel velocities.
A rotation matrix-based formulation would typically include a full transformation from body-frame velocities $[v_x \quad v_y \quad \omega]^T$ to world-frame velocities:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

But the differential drive robot has **non-holonomic constraints**—it cannot move sideways ($v_y = 0$). Therefore, this general transformation is not applicable because:

- The lateral velocity component v_y is always zero.
- The system is underactuated and constrained to planar trajectories that follow the heading direction only.



Inverse Kinematics: Converting desired robot velocities to wheel velocities:

$$\omega_L = \frac{2v - \omega L}{2r} \quad (5.1)$$
$$\omega_R = \frac{2v + \omega L}{2r} \quad (5.2)$$

Proof

To find ω_L and ω_R in terms of the desired robot velocities (v and ω), we need to solve the forward kinematics equations for these variables.
Step 1: Set up the equations
We have:

$$v = \frac{r \cdot \omega_L + r \cdot \omega_R}{2} \tag{P1.1}$$

$$\omega = \frac{r \cdot \omega_R - r \cdot \omega_L}{L} \tag{P1.2}$$

Step 2: Rearrange Eqn. (P1.1)
Multiply both sides by 2:

$$2v = r \cdot (\omega_R + \omega_L) \tag{P1.3}$$

Step 3: Rearrange Eqn. (P1.2)
Multiply both sides by L:

$$\omega L = r \cdot (\omega_R - \omega_L) \tag{P1.4}$$

Step 4: Create a system of two equations
We now have:

$$\omega_L + \omega_R = \frac{2v}{r} \tag{P1.5}$$

$$(\omega_R - \omega_L) = \frac{\omega L}{r} \tag{P1.6}$$

Step 5: Solve for $\omega_L + \omega_R$ and $\omega_R - \omega_L$

Step 6: Solve for ω_R
Add the two equations ((P1.5) and (P1.6)):

$$2\omega_R = \frac{2v}{r} + \frac{\omega L}{r} \tag{P1.7}$$

$$\omega_R = \left(\frac{2v + \omega L}{2r}\right) \tag{P1.8}$$

Step 7: Solve for ω_L
Substitute ω_R in Eqn. (P1.8) into Eqn. (P1.5)

$$\omega_L + \left(\frac{2v + \omega L}{2r}\right) = \frac{2v}{r} \tag{P1.9.1}$$

$$\omega_L = \frac{2v}{r} - \left(\frac{2v + \omega L}{2r}\right) \tag{P1.9.2}$$

$$\omega_L = \frac{4v}{2r} - \left(\frac{2v + \omega L}{2r}\right) \tag{P1.9.3}$$

$$\omega_L = \left(\frac{2v - \omega L}{2r}\right) \tag{P1.10}$$

End of Proof

Key Assumptions

1. **Ideal Rolling Contact**: The wheels roll without slipping on the surface. This is a critical assumption that allows us to relate wheel rotation directly to robot motion.
2. **Rigid Body**: The robot is assumed to be a rigid body with no flexibility between components.
3. **Planar Motion**: The robot moves on a flat, horizontal surface with no inclines or irregularities.

4. **Point Contact**: The wheels make point contact with the ground.
5. **No Lateral Slip:** The wheels cannot slide sideways (non-holonomic constraint).
6. **Constant Parameters**: The wheel radius (r) and wheel base (L) are constant.
7. **Symmetric Design**: The robot is assumed to be symmetric around its center axis.
8. **No Friction Model**: The equations assume perfect traction without modeling complex friction.
9. **Instantaneous Velocity Changes**: The kinematic model allows for instantaneous changes in velocity, unlike a dynamic model which would account for acceleration limits.

2.1.2 Dynamic Model of Differential Drive Robot

The dynamic model considers forces, torques, and inertial properties.

Additional Parameters:

- Mass: m of the robot
- Moment of inertia: Iaround the center
- Friction coefficients: B_R, B_L for wheels

Dynamic Equations via Lagrangian Method

The Lagrangian $L = T - V$, where T is kinetic energy and V is potential energy.

For a differential drive robot on a flat plane (ignoring potential energy):

$$T = \frac{1}{2}mv^2 + \frac{1}{2}I_z\omega^2 \tag{6}$$

The equations of motion are derived using:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = Q_i \tag{7}$$

where q_i are the generalized coordinates (x, y, θ) and Q_i are the generalized forces.

For a differential drive robot:

- Calculate the total kinetic energy:
$$T = \frac{1}{2}mv^2 + \frac{1}{2}I_z\omega^2 \tag{8}$$
- Express in terms of wheel velocities:
$$v = \frac{r\cdot(\omega_R + \omega_L)}{2} \tag{9.1}$$
$$\omega = \frac{r\cdot(\omega_R - \omega_L)}{L} \tag{9.2}$$
- Substituting Eqn. (9.1) and (9.2) into Eqn. (8)
$$T = \frac{1}{2}m\left(\frac{r\cdot(\omega_R + \omega_L)}{2}\right)^2 + \frac{1}{2}I_z\left(\frac{r\cdot(\omega_R - \omega_L)}{L}\right)^2 \tag{10}$$
- Apply Lagrangian equations for wheel rotations: $\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\omega}_R}\right) = \tau_R$(torque on right wheel)$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\omega}_L}\right) = \tau_L$ (torque on left wheel)
- The final dynamic equations relate wheel torques to accelerations:

$$\tau_R = \left(\frac{mr^2}{4} + \frac{I_z r^2}{L^2}\right)\dot{\omega}_R + \left(\frac{mr^2}{4} - \frac{I_z r^2}{L^2}\right)\dot{\omega}_L \quad (11.1)$$
$$\tau_L = \left(\frac{mr^2}{4} - \frac{I_z r^2}{L^2}\right)\dot{\omega}_R + \left(\frac{mr^2}{4} + \frac{I_z r^2}{L^2}\right)\dot{\omega}_L \quad (11.2)$$

$$\begin{bmatrix}\tau_R\\ \tau_L\end{bmatrix} = \begin{bmatrix}\frac{mr^2}{4} + \frac{I_z r^2}{L^2} & \frac{mr^2}{4} - \frac{I_z r^2}{L^2}\\ \frac{mr^2}{4} - \frac{I_z r^2}{L^2} & \frac{mr^2}{4} + \frac{I_z r^2}{L^2}\end{bmatrix}\begin{bmatrix}\dot{\omega}_R\\ \dot{\omega}_L\end{bmatrix} + \begin{bmatrix}B_R & 0\\ 0 & B_L\end{bmatrix}\begin{bmatrix}\omega_R\\ \omega_L\end{bmatrix} \quad (12)$$

Dynamic Equations: Relating torque inputs to wheel acceleration:

$$\begin{bmatrix}\tau_R\\ \tau_L\end{bmatrix} = \begin{bmatrix}J_R & J_{RL}\\ J_{RL} & J_L\end{bmatrix}\begin{bmatrix}\dot{\omega}_R\\ \dot{\omega}_L\end{bmatrix} + \begin{bmatrix}B_R & 0\\ 0 & B_L\end{bmatrix}\begin{bmatrix}\omega_R\\ \omega_L\end{bmatrix} \quad (13)$$

Where:

- $J_R = \frac{mr^2}{4} + \frac{I_z r^2}{L^2} = J_L$re the effective inertias for each wheel
- $J_{RL} = \frac{mr^2}{4} - \frac{I_z r^2}{L^2}$ is the coupling inertia
- τ_R, τ_L are the applied torques
- B_R, B_L are the viscous friction coefficients

2.2 Simulation Implementation (simulation_mobile_robot_with_obstacles.m)

2.2.1 State Initialization

```
% Initialize state variables
x = 0;        % initial x position [m]
y = 0;        % initial y position [m]
theta = 0;     % initial orientation [rad]
omega_r = 0;   % initial right wheel angular velocity [rad/s]
omega_l = 0;   % initial left wheel angular velocity [rad/s]
```

2.2.2 Dynamic Update Function

The dynamics equations are implemented as:

```
function [omega_r, omega_l] = update_dynamics(omega_r, omega_l, tau_r, tau_l, robot,
dt)
   % Compute the determinant of the inertia matrix
   det_J = robot.J_r * robot.J_l - robot.J_rl^2;

   % Compute wheel accelerations
   omega_dot_r = ((robot.J_l * (tau_r - robot.B_r * omega_r) - ...
            robot.J_rl * (tau_l - robot.B_l * omega_l)) / det_J);

   omega_dot_l = ((robot.J_r * (tau_l - robot.B_l * omega_l) - ...
            robot.J_rl * (tau_r - robot.B_r * omega_r)) / det_J);
```

```
  % Update wheel velocities
  omega_r = omega_r + omega_dot_r * dt;
  omega_l = omega_l + omega_dot_l * dt;
end
```

2.2.3 Forward Kinematics Implementation

```
function [v, omega] = forward_kinematics(omega_r, omega_l, robot)
  % Linear and angular velocity of the robot
  v = robot.wheel_radius * (omega_r + omega_l) / 2;
  omega = robot.wheel_radius * (omega_r - omega_l) / robot.wheel_base;
end
```

2.2.4 Kinematic Update Function

```
function [x, y, theta] = update_kinematics(x, y, theta, v, omega, dt)
  % Update robot position and orientation
  x = x + v * cos(theta) * dt;
  y = y + v * sin(theta) * dt;
  theta = theta + omega * dt;

  % Normalize theta to [-pi, pi]
  theta = atan2(sin(theta), cos(theta));
end
```

2.2.5 Simulation Loop Structure

```
% Main simulation loop
for i = 1:num_steps
  % 1. Calculate control inputs (torques)

  % 2. Update dynamics (wheel velocities)
  [omega_r, omega_l] = update_dynamics(omega_r, omega_l, tau_r, tau_l, robot, dt);

  % 3. Calculate robot velocities from wheel velocities
  [v, omega] = forward_kinematics(omega_r, omega_l, robot);

  % 4. Update robot position and orientation
  [x, y, theta] = update_kinematics(x, y, theta, v, omega, dt);

  % 5. Store states for later analysis
  history.x(i+1) = x;
  history.y(i+1) = y;
  history.theta(i+1) = theta;
end
```

2.2.6 Position Control Implementations

The simulation implements three distinct position control scenarios, each demonstrating different approaches to manoeuvring the robot along specified paths:

Scenario 1: Simple Motion Sequence

This scenario demonstrates basic open-loop control through a sequence of predefined torque commands:

```
% Motion sequence implementation
for i = 1:num_steps
  % Determine control inputs based on the current stage
  if i < num_steps/3
    % Stage 1: Circular motion through differential torque
    tau_r = 0.3;
    tau_l = 0.2;
  elseif i < 2*num_steps/3
    % Stage 2: Straight motion with equal torques
    tau_r = 0.3;
    tau_l = 0.3;
  else
    % Stage 3: Turn right with reduced right wheel torque
    tau_r = 0.1;
    tau_l = 0.3;
  end

  % System update and state recording
  [omega_r, omega_l] = update_dynamics(omega_r, omega_l, tau_r, tau_l, robot, dt);
  [v, omega] = forward_kinematics(omega_r, omega_l, robot);
  [x, y, theta] = update_kinematics(x, y, theta, v, omega, dt);
end
```

Key characteristics:
- Time-based switching between control modes
- Open-loop control with no feedback from position or orientation
- Demonstrates basic maneuvers: circular motion, straight-line motion, and turning
----

Scenario 2: Figure-8 Path Following
This scenario implements closed-loop position control to follow a parametric figure-8 path:

```
% Path definition
t_values = linspace(0, 2*pi, num_steps+1);
path_x = scale_x * sin(t_values);
path_y = scale_y * sin(t_values) .* cos(t_values);

% Control loop
for i = 1:num_steps
  % Calculate position and heading errors
  dx = target_x - current_x;
  dy = target_y - current_y;
  distance = sqrt(dx^2 + dy^2);

  % Calculate target heading using look-ahead for smoother path
  if i < num_steps
    target_heading = atan2(path_y(i+1) - path_y(i), path_x(i+1) - path_x(i));
  else
    target_heading = atan2(dy, dx);
  end

  % Calculate heading error and normalize to [-pi, pi]
  heading_error = target_heading - current_theta;
  heading_error = atan2(sin(heading_error), cos(heading_error));

  % Position controller: proportional control with saturation
  v = kp_v * distance;  % Linear velocity proportional to distance error
  omega = kp_omega * heading_error;  % Angular velocity proportional to heading error

  % Limit control outputs
  v = max(min(v, 1.0), -1.0);
  omega = max(min(omega, 2.0), -2.0);

  % Convert to wheel velocities and then torques
  [omega_l, omega_r] = inverse_kinematics(v, omega, robot);
  tau_l = omega_l * 0.1;  % Simple conversion factor
  tau_r = omega_r * 0.1;

  % System update and state recording
  [omega_r, omega_l] = update_dynamics(omega_r, omega_l, tau_r, tau_l, robot, dt);
  [v, omega] = forward_kinematics(omega_r, omega_l, robot);
  [x, y, theta] = update_kinematics(x, y, theta, v, omega, dt);
end
```

Key characteristics:

- Closed-loop proportional control with position and heading feedback
- Look-ahead targeting for smoother path following
- Error minimization in both distance and orientation
- Control signal limiting to respect robot constraints

Scenario 3: Kinematics vs Dynamics Comparison

This scenario demonstrates the differences in position control between pure kinematic models and full dynamic models:

```
% Input: sudden acceleration then sudden stop
if i < num_steps_comp/3
   % Accelerate phase
   tau_l = 0.3;
   tau_r = 0.3;

   % For kinematic model, directly set velocities (instantaneous)
   v_kin = 0.5;      % Direct velocity control
   omega_kin = 0.0;  % No rotation
else
   % Sudden stop phase
   tau_l = 0.0;
   tau_r = 0.0;

   % For kinematic model, instantaneous stop
   v_kin = 0.0;
   omega_kin = 0.0;
end

% Update dynamics model (realistic physics)
[omega_r_dyn, omega_l_dyn] = update_dynamics(omega_r_dyn, omega_l_dyn, tau_r, tau_l, robot_dyn, dt);
[v_dyn, omega_dyn] = forward_kinematics(omega_r_dyn, omega_l_dyn, robot_dyn);
[x_dyn, y_dyn, theta_dyn] = update_kinematics(x_dyn, y_dyn, theta_dyn, v_dyn, omega_dyn, dt);

% Update kinematic model (ideal behavior)
[x_kin, y_kin, theta_kin] = update_kinematics(x_kin, y_kin, theta_kin, v_kin, omega_kin, dt);
```

Key observations:
- The kinematic model exhibits instantaneous velocity changes, while the dynamic model shows realistic acceleration and deceleration profiles due to inertia
- The dynamic model reveals physical limitations in position control that aren't captured by pure kinematics

- Higher mass and friction in the dynamic model result in slower response and longer stopping distances
- Path deviation occurs in dynamic models during rapid manoeuvres due to inertial effects

----

2.2.7 Position Control Analysis

The implemented position control strategies reveal several key insights:

1. Open-loop Control Limitations:
 • Scenario 1 demonstrates that open-loop control (fixed torque sequences) can achieve basic maneuvers but lacks precision for complex path following
 • Position drift accumulates over time due to uncompensated dynamics and disturbances

2. Closed-loop Control Advantages:
 • Position feedback in Scenario 2 enables the robot to continuously correct its path
 • Proportional control on both linear and angular velocities effectively minimizes position and orientation errors
 • Look-ahead targeting improves path following smoothness compared to pure position error minimization

3. Dynamic Model Considerations:
 • Scenario 3 demonstrates that realistic dynamic effects (inertia, friction) significantly impact position control performance
 • Controllers designed using only kinematic models may perform poorly on real robots with significant mass and inertia
 • Stopping and direction changes require torque planning that accounts for dynamic effects

4. Control Parameter Effects:
 • Higher proportional gains yield faster error reduction but can cause oscillations around the target path
 • Velocity and acceleration limitations (saturation) are necessary for stable control and realistic robot behavior
 • The controller must balance path following accuracy with smooth motion profiles

These scenarios provide a foundation for more advanced control techniques such as model predictive control or adaptive control that could further improve path following performance while accounting for dynamic constraints.

3. Discussion and Exercises

3.1 Theoretical Questions

1. What are the fundamental differences between a kinematic and dynamic model?
2. Explain how the coupling term J_{RL} affects the motion of a differential drive robot.
3. What are the limitations of both simulation models? How could they be improved?

3.2 Programming Exercises

1. Modify the differential drive simulation to include wheel slip

4. Conclusion

This module has demonstrated the progression from theoretical equations to practical simulations for robotic systems. By understanding both the mathematical foundations and their implementation, students gain insight into the behavior of complex robotic systems and develop the skills to design, analyze, and control them.

The relationship between theory and simulation is bidirectional - theoretical models inform simulation design, while simulation results can validate or challenge our theoretical understanding. This iterative process is central to the advancement of robotics and control systems.