Robotics Programming Laboratory

Bertrand Meyer
Jiwon Shin

Lecture 3: Robot Control
Go forward, go right

Holonomic
DDOF = DOF

Nonholonomic
DDOF < DOF

DOF: Ability to achieve various poses
DDOF: Ability to achieve various velocities
Differential drive

Forward: $\dot{\varphi}_L = \dot{\varphi}_R > 0$
Backward: $\dot{\varphi}_L = \dot{\varphi}_R < 0$
Right turn: $\dot{\varphi}_L > \dot{\varphi}_R$
Left turn: $\dot{\varphi}_L < \dot{\varphi}_R$
Differential drive

Input: \((v, \omega)\)

\[
\begin{align*}
\dot{x} &= v \cos \theta \\
\dot{y} &= v \sin \theta \\
\dot{\theta} &= \omega
\end{align*}
\]
Differential drive

\[ \dot{x} = R \frac{\dot{\varphi}_L + \dot{\varphi}_R}{2} \cos \theta \]

\[ \dot{y} = R \frac{\dot{\varphi}_L + \dot{\varphi}_R}{2} \sin \theta \]

\[ \dot{\theta} = \frac{R}{B} (\dot{\varphi}_R - \dot{\varphi}_L) \]
Odometry: intuition
Odometry for small $t$

$x(t) = x(t-1) + d_C \cos \theta(t)$

$y(t) = y(t-1) + d_C \sin \theta(t)$

$\theta(t) = \theta(t-1) + \theta_C$

$d_C = \frac{1}{2} (d_L + d_R)$

$\theta_C = \frac{d_R - d_L}{B}$
More accurate odometry for small $t$

\[ d_C = \frac{1}{2} (d_L + d_R) \]

\[ \theta_C = \arctan\left(\frac{d_R - d_L}{B}\right) \]

\[ x(t) = x(t-1) + d_C \cos(\theta(t-1) + \frac{1}{2} \theta_C) \]

\[ y(t) = y(t-1) + d_C \sin(\theta(t-1) + \frac{1}{2} \theta_C) \]

\[ \theta(t) = \theta(t-1) + \theta_C \]
Wheel encoder

How do we get the distance each wheel has moved?

- If the wheel has $N$ ticks per revolution:

\[
\Delta n_{\text{tick}} = n_{\text{tick}}(t) - n_{\text{tick}}(t - 1)
\]

\[
d = 2\pi R \frac{\Delta n_{\text{tick}}}{N}.
\]

- Thymio: $d = d \Delta t$

DRIFT
Go to goal

Goal

Control
Feedback

A collection of two or more dynamical systems, in which each system influences the other, resulting in strongly-coupled dynamics.

- **Open loop**: the systems are not interconnected (no feedback)

- **Closed loop**: the systems are interconnected (with feedback)
Control

The use of algorithms and feedback in engineered systems

Robot speed control

- **Actuator**: set the robot’s speed
- **Sensor**: sense the robot’s actual speed
- **Control goals**: set the robot’s speed such that:
  - **Stability**: the robot maintains the desired speed
  - **Performance**: the robot responds quickly to changes
  - **Robustness**: the robot tolerates perturbation in dynamics
On-off controller

\[
u = \begin{cases} 
  u_{\text{max}} & \text{if } e > 0 \\
  u_{\text{min}} & \text{if } e < 0 
\end{cases}
\]

\[
\text{error} := \text{set\_point} - \text{measured}
\]

\[
\text{if } \text{error} > 0.0 \text{ then}
\]

\[
\text{output} := \text{max}
\]

\[
\text{else}
\]

\[
\text{if } \text{error} < 0.0 \text{ then}
\]

\[
\text{output} := \text{min}
\]

\[
\text{end}
\]

\[
\text{end}
\]
On-off controller

\[ u = \begin{cases} 
    u_{\text{max}} & \text{if } e > 0 \\
    u_{\text{min}} & \text{if } e < 0 
\end{cases} \]
Proportional controller

\[ u(t) = k_p e(t) \]

error := set_point - measured
output := k_p * error
Proportional controller

\[ u(t) = k_p e(t) \]
Proportional derivative controller

\[ u(t) = k_p e(t) + k_d \frac{de(t)}{dt} \]

error := set_point - measured
proportional := k_p * error
derivative := k_d * (error - prev_error)/dt
output := proportional + derivative
Proportional derivative controller

\[ u(t) = k_p e(t) + k_d \frac{de(t)}{dt} \]
Proportional integral derivative controller

\[ u(t) = k_p e(t) + k_i \int_0^t e(\tau) d\tau + k_d \frac{de(t)}{dt} \]

\begin{align*}
\text{error} & := \text{set\_point} - \text{measured} \\
\text{proportional} & := k_p \times \text{error} \\
\text{integral} & := k_i \times (\text{accumulated\_error} + \text{error} \times dt) \\
\text{derivative} & := k_d \times (\text{error} - \text{prev\_error})/dt \\
\text{output} & := \text{proportional} + \text{integral} + \text{derivative}
\end{align*}
Proportional integral derivative controller

\[ u(t) = k_p e(t) + k_i \int_0^t e(\tau) \, d\tau + k_d \frac{de(t)}{dt} \]
Go to goal

error := $\theta_{goal} - \theta_{current}$
Control gains
Control gains

Ziegler-Nicols method

- Set $K_i$ and $K_d$ to 0.
- Increase $K_p$ until $K_u$ at which point the output starts to oscillate.
- Use $K_u$ and the oscillation period $T_u$ to set the control gains.

<table>
<thead>
<tr>
<th>Control Type</th>
<th>$K_p$</th>
<th>$K_i$</th>
<th>$K_d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>$0.50K_u$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PI</td>
<td>$0.45K_u$</td>
<td>$1.2K_p/T_u$</td>
<td>-</td>
</tr>
<tr>
<td>PID</td>
<td>$0.60K_u$</td>
<td>$2K_p/T_u$</td>
<td>$K_pT_u/8$</td>
</tr>
</tbody>
</table>

Manual tuning!
P, PI, PID, ….?

\[ u(t) = k_p e(t) + k_i \int_0^t e(\tau) d\tau + k_d \frac{de(t)}{dt} \]

\[ k_p, k_i, k_d \neq 0 \]

\[ k_i = 0 \]

\[ k_d = 0 \]

\[ k_p = 0 \]
Software engineering tips

- Does functionality F belong to class C? In its own class?
- Can functionality F be generalized?
- Is number N a constant? A variable?
- Should number N be in the source code? Command line input? Read from a file?