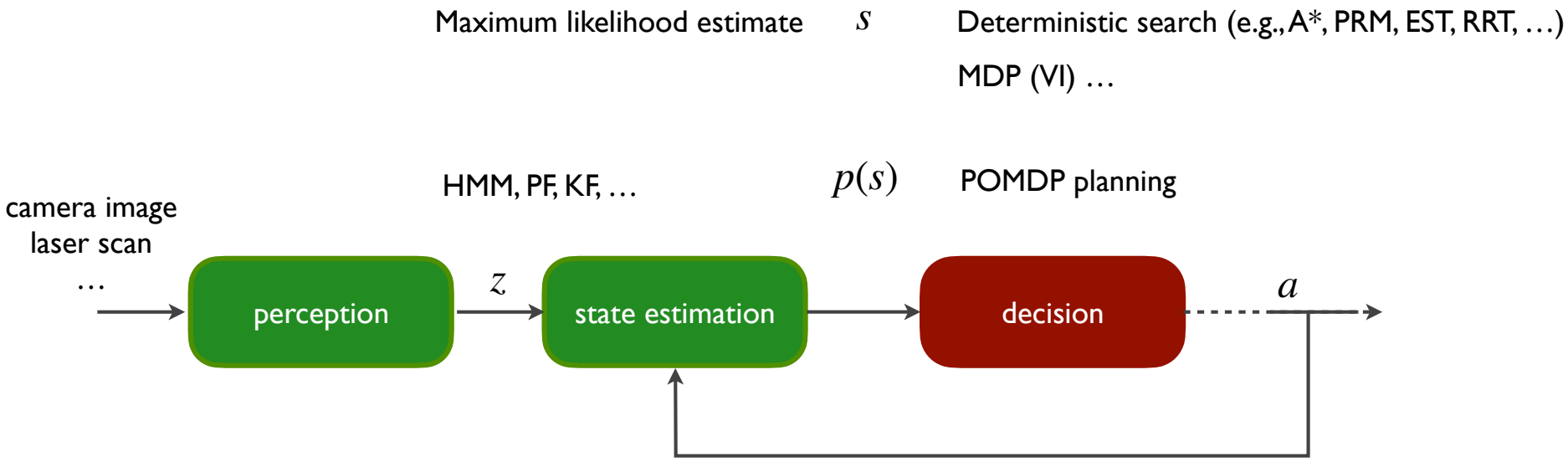
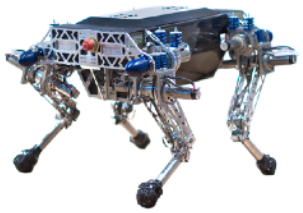
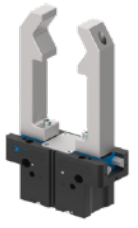
Integrating perception, state estimation, and decision. State estimation is the input to the decision module. Accurate state estimation contributes to good decision making and vice versa. Good decision making may also help state estimation by acting to acquire informative observations.

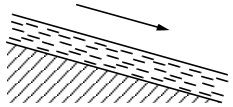
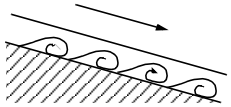
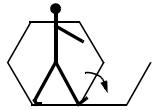



Very briefly, what does the human do?

Move on the ground.

Type of motion	Resistance to motion	Basic kinematics of motion
Flow in a Channel 	Hydrodynamic forces	Eddies 
Crawl 	Friction forces	Longitudinal vibration 
Sliding 	Friction forces	Transverse vibration 
Running 	Loss of kinetic energy	Periodic bouncing on a spring 
Walking 	Loss of kinetic energy	Rolling of a polygon (see figure 2.2) 

Question. What a common motion mode that is missing in nature?

Wheels

Legs.

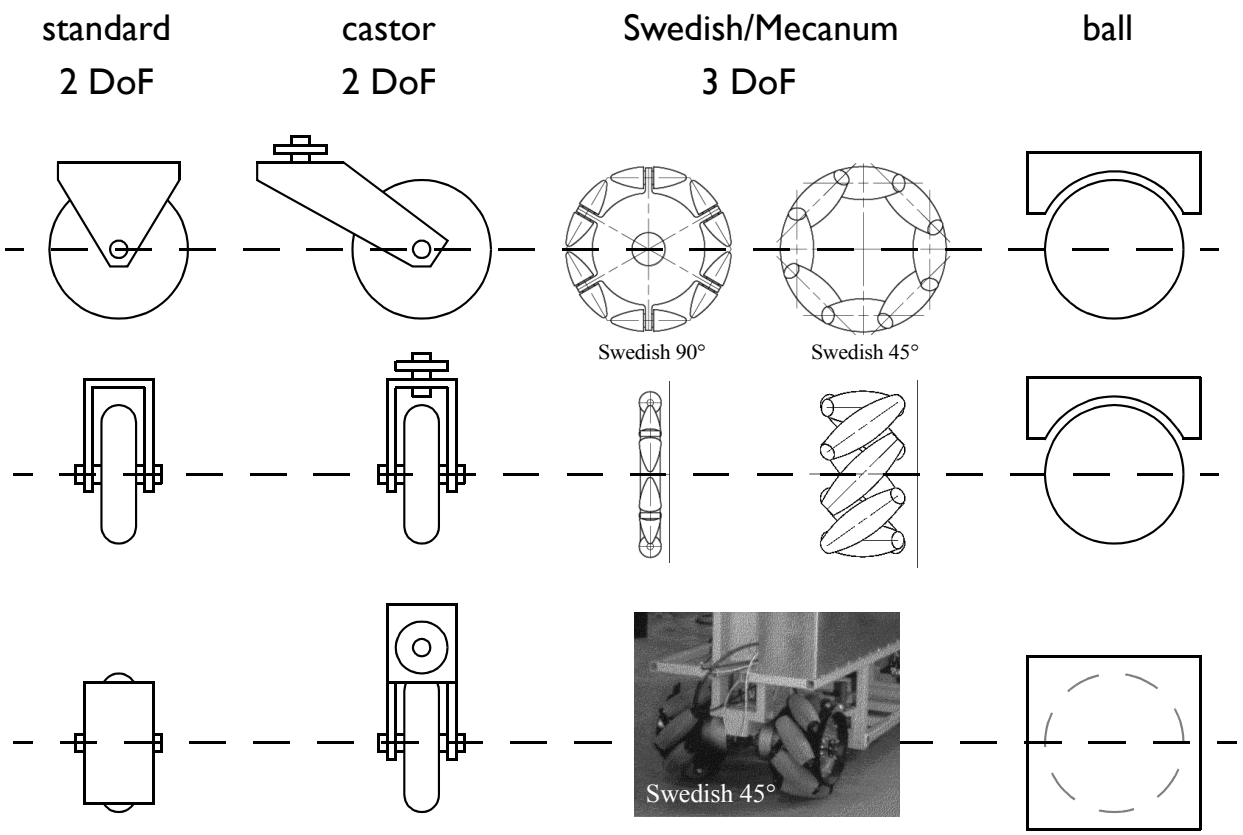
One-legged hopper, MIT, 1983

Legs.

Big dog, Boston Dynamics

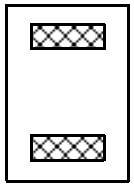
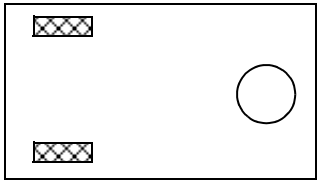
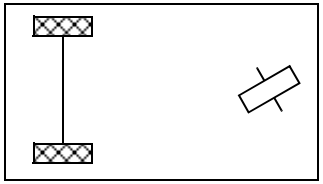
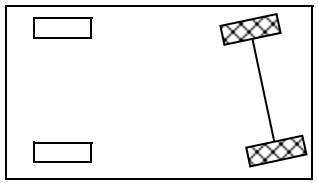
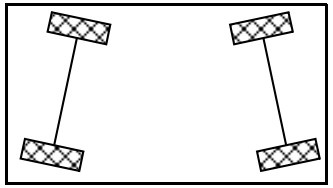
Legs.

Wheels.

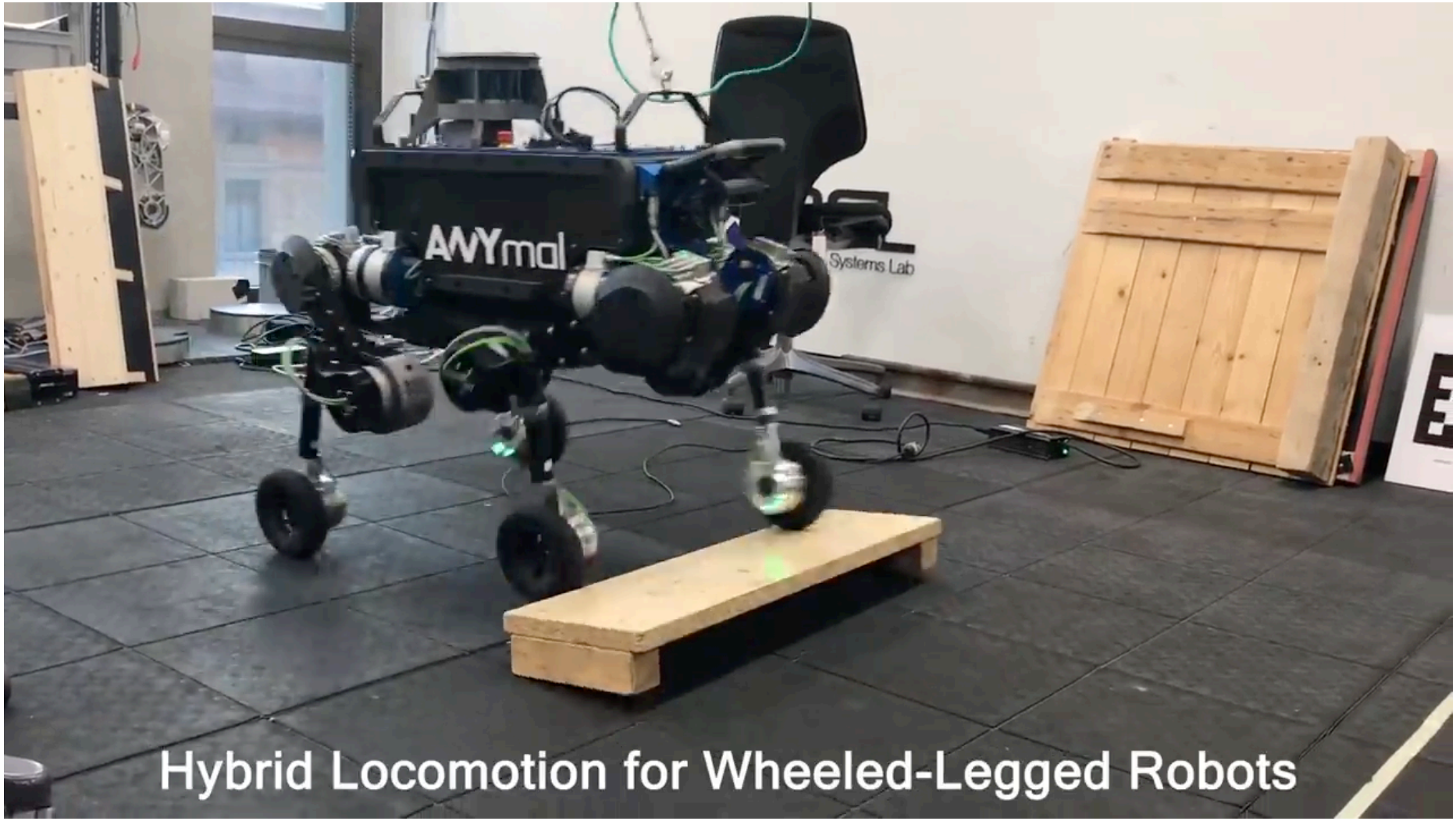

standard
2 DoF
castor
2 DoF
Swedish/Mecanum
3 DoF
ball
Swedish 90°
Swedish 45°
Swedish 45°


OmniTurtle

Wheels.

	One steering wheel in the front, one traction wheel in the rear.	Bicycle, motorcycle.
	Two differential-drive traction wheels.	Segway
	Two independently-driven wheels in the rear/front, one unpowered omnidirectional wheel in the front/rear.	Many indoor robots.
	Two connected traction wheels in rear, one steered free wheel in the front.	Piaggio minitrucks.
	Two motorized, steered wheels in the front, two free wheels in the rear.	Front-wheel-drive cars.
	Four steered, motorized wheels	Four-wheel-drive cars.

Legged-wheels.



Tracks.

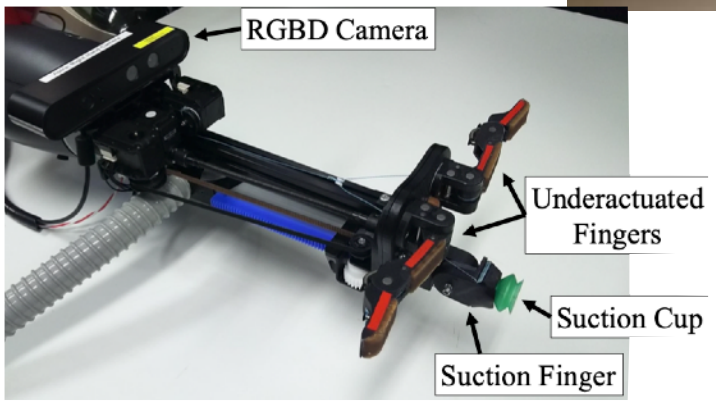
Move in the air.

Helicopters can hover in midair.

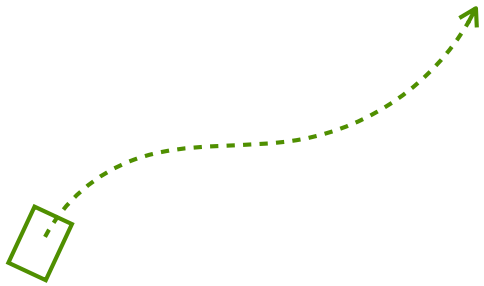
Move in the sea.

Underwater gliders use buoyancy rather conventional propeller.

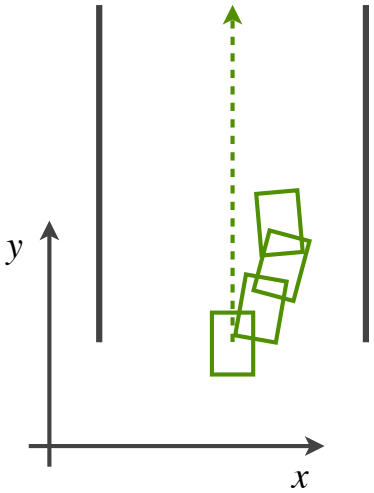
Arms and hands.



Motion control. After planning a path, how does the robot actually execute it?



Consider the simplified setting: keep the robot moving forward along a straight path. The robot may deviate from the nominal path because of uneven ground, friction, wheel mechanical imperfection, …. To start, we control only the lateral motion through the lateral velocity u_x .

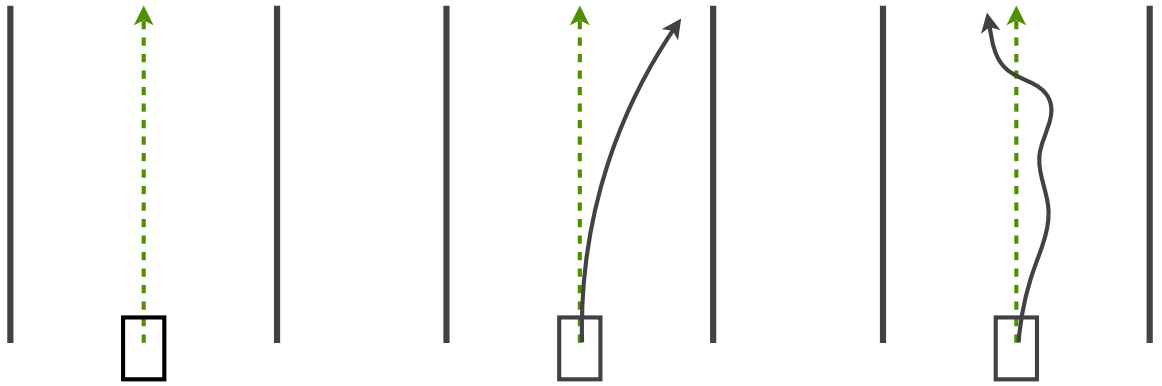


lateral velocity *longitudinal velocity*

velocity control $u = (u_x, u_y)$

Open-loop control. Open-loop control does not consider the resulting state of a control action. It assumes that control actions achieve their intended effect perfectly, without any disturbance. It does not receive or require any feedback.

To move along the path, open-loop control sets $u_x = 0$ and u_y to a constant speed.



The assumption of perfect execution often fails in practice. The ground may be tilted to one side, the wheels may be slightly misaligned, … Any of these may cause the robot to veer off the nominal path unrecoverably.

Sometimes, the random errors may "cancel off" on the average by chance, but there is no assurance.

Close-loop (feedback) control. Feedback from sensor measurements are important to correct the error in control action execution.

Propotional control (P-control). Suppose that

$$e(t) = x_{\mathrm{d}}(t) - x(t)$$

It would be natural to make the control proportional to the size of the error: the bigger the error, the greater the correction:

$$u_x(t) = K_{\mathrm{p}} e(t)$$

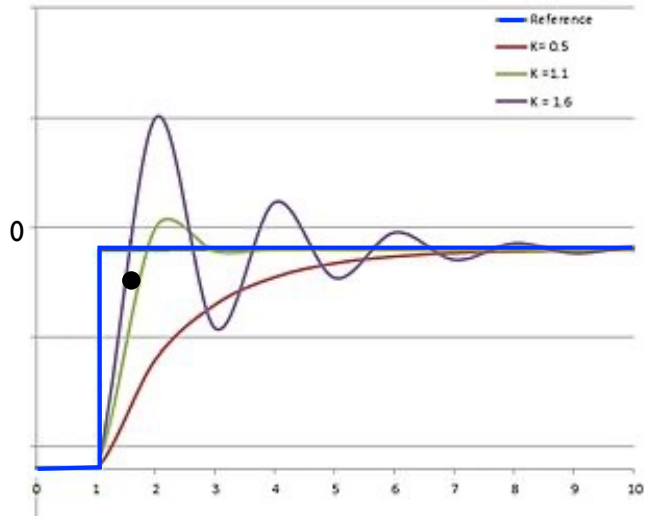
So we have

$$\dot{x} = u_x$$
$$\dot{e} = -K_{\mathrm{p}} e$$

Solving the above equation, we have,

$$e(t) = e_0 \exp(-K_{\mathrm{p}} t)$$

which goes to 0 over time. If K_{p} is large, the error approaches 0 faster, resulting in faster response time and a more sensitive system. However, this requires continuous control u_x. In digital systems, we apply fixed constant u_x for a small time duration Δt. If K_{p} is large, we may "overshoot".



Suppose that the robot navigates through a stretch of surface banking to one side. Now,

$$\dot{x} = u_x + \delta$$ ← *lateral velocity disturbance*

$$\dot{e} = -K_p e - \delta$$

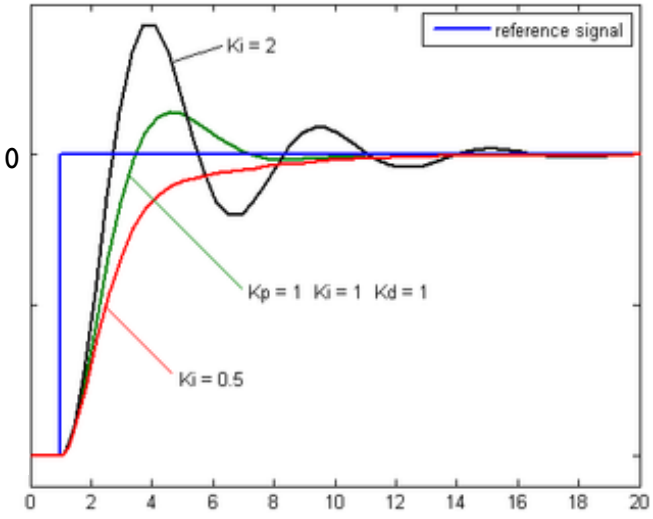
Solving the above equation, we have $e(t) = -\dfrac{e_0 \exp(-K_p t)}{K_p} - \dfrac{\delta}{K_p}$.

For any finite value of K_p, the error δ / K_p remains even as $t \to \infty$. Larger values of K_p help to reduce this error, but do not eliminate this steady-state error. P-control cannot eliminate this error, as it results from the cumulative effect of disturbance over time.

Proportional integral control (PI control). We add a integral term to eliminate the cumulative error in the steady state:

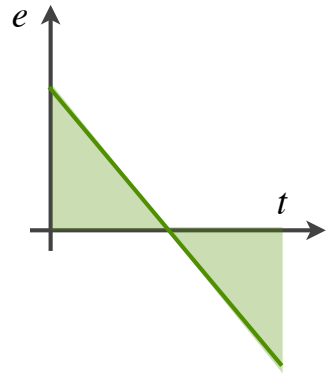
$$u_x(t) = K_p e(t) + K_i \int_{t_0}^{t} e(\tau)\, d\tau$$

The integral term is proportional to the cumulative error over a small current time interval $[t_0, t]$. If the error persists, the cumulative error increases, resulting in greater intervention from the control.



Question. Can we use the integral control alone without the proportional term?

The cumulative error has greater "momentum". It is slow to start and slow to end. This makes it even more likely to "overshoot".



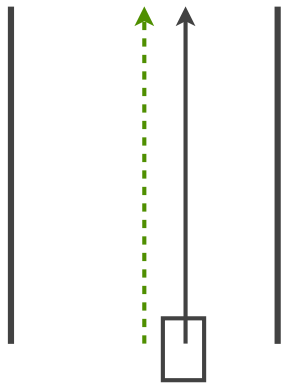
At time t, the error is substantial, but the signed cumulative error is 0. So the integral control is 0 as well. This is clearly undesirable, as the error at time t remains large.

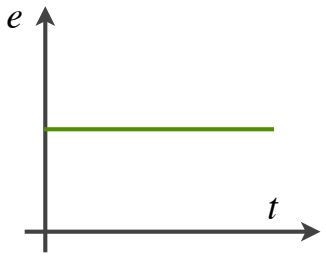
Proportional integral derivative control (PID control). We add a derivative term to counteract overshooting. The applied control action depends on the rate of change in error. If the robot is getting close to the desired goal, large control action is unsuitable: it may overshoot the goal.

$$u_x = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(\tau)\,\mathrm{d}\tau + K_\mathrm{d} \dot{e}$$

error — cumulative error — rate of change in error

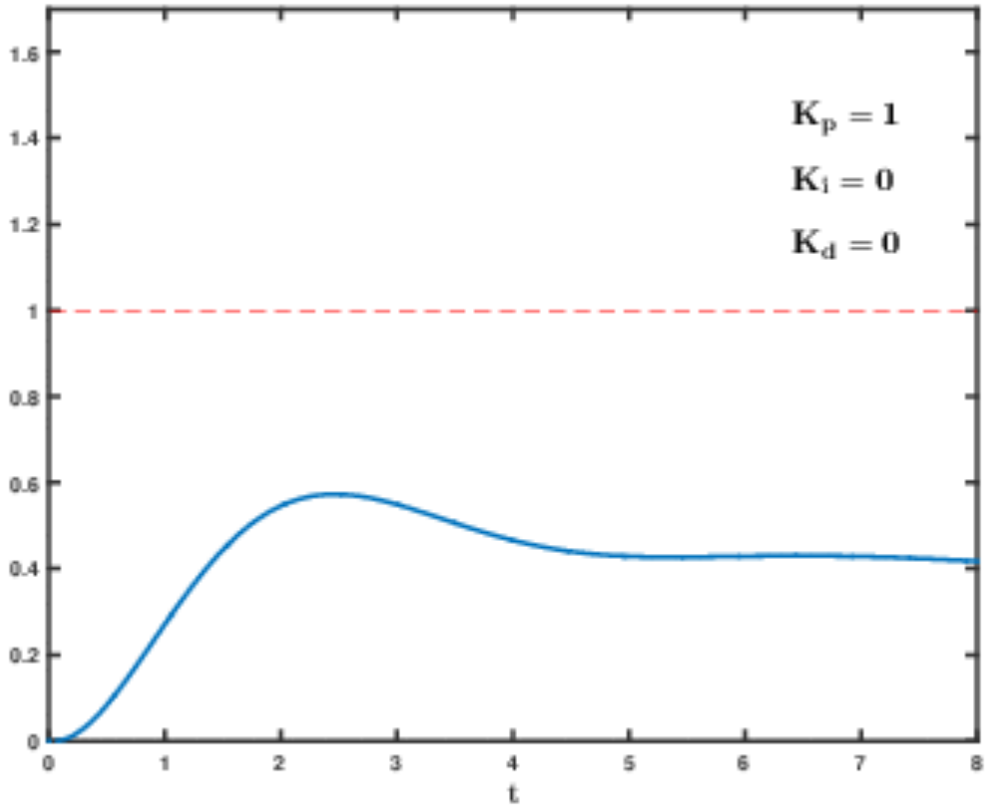
Question. Can we use the derivative control alone to keep the robot move along the centerline?

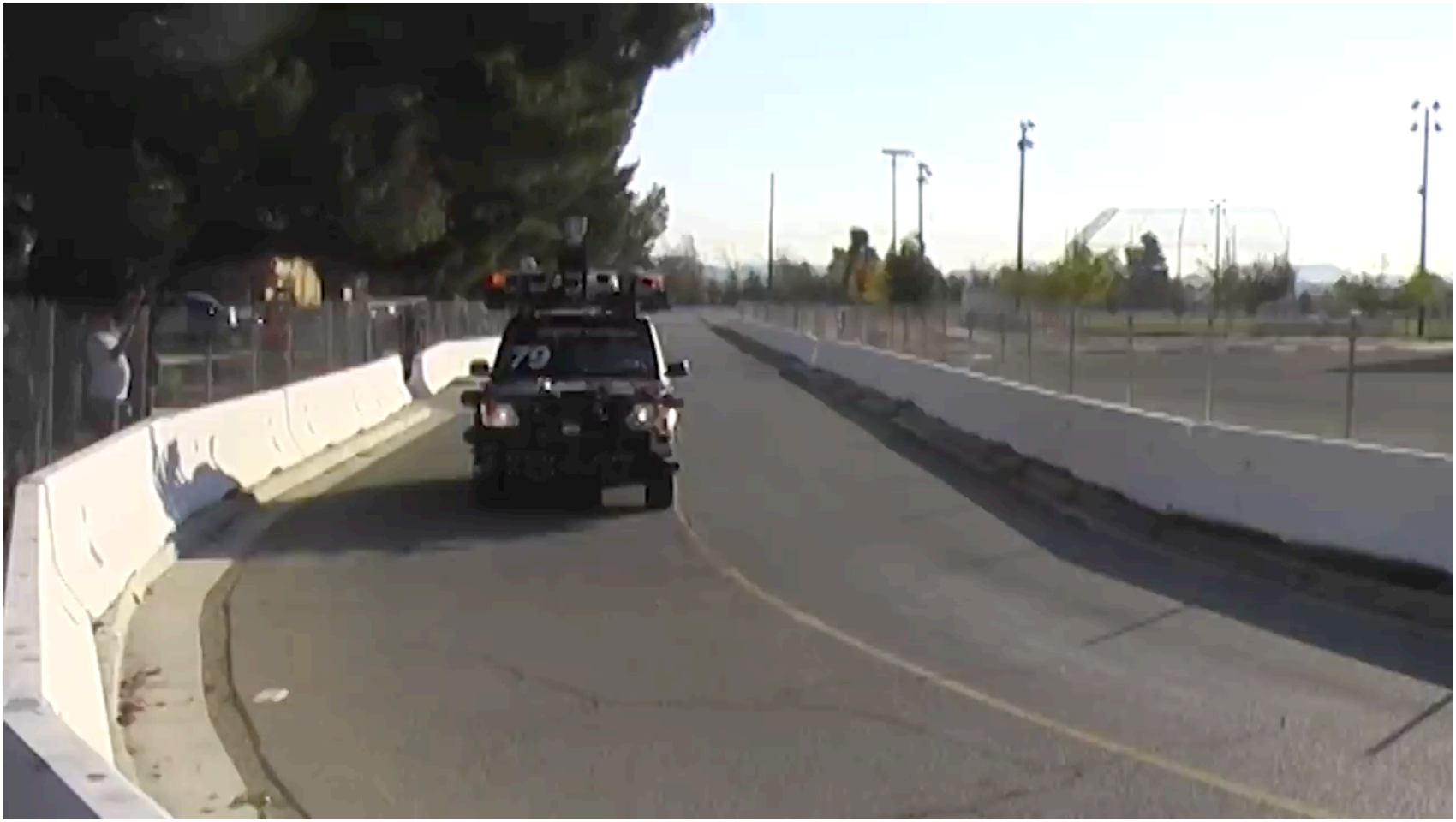




If the error is constant, the derivative control is 0.

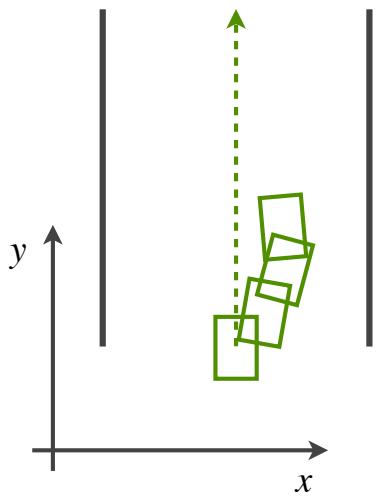
Now we must tune the control gain parameters K_p, K_i, and K_d.
There are various heuristics, e.g., Ziegler–Nichols method.





http://youtube.com/watch?v=4Y7zG48uHRo

General trajectory control.



Now we understand how to control u_x to keep the robot deviate from the center. The same idea applies to the longitudinal control u_y by setting

$$e(t) = y_{\mathrm{d}}(t) - y(t)$$

In fact, $(x_d(t), y_d(t))$ may represent any arbitrary trajectory and not necessarily a straight one.

All the controllers discussed so far choose the control action according the resulting error, but do not consider the specific system under control. Controlling a ground robot is no different from controlling a helicopter, other than different values for K_{p}, K_{i}, or K_{d}.

Naturally, by taking into account specific system properties, we should be able to derive better control. Further, we may be able to derive **optimal control**.

Linear quadratic regulator (LQR). LQR assumes that the system dynamics is **linear** and the cost is **quadratic**.

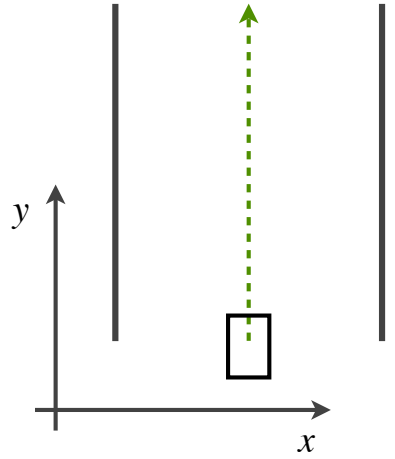
The linear discrete state-transition equation is given by

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t \quad \text{for } t = 0,1,\ldots,N.$$

($\mathbf{x}_t$: vector; A: matrix)

Example. Consider again the example of keeping the robot moving along the a straight path:

- $\mathbf{x}_t = [x_t, v_t]^T$ for lateral position x_t, lateral velocity v_t.
- Apply lateral force $\mathbf{u}_t$.
- Lateral friction force is $-\eta v_t$.
- The robot has mass m.



By Newton's law,

$$ma_t = u_t - \eta v_t$$

For small time duration Δt,

$$m\frac{v_{t+1} - v_t}{\Delta t} \approx u_t - \eta v_t$$

$$v_t \approx \frac{x_{t+1} - x_t}{\Delta t}$$

We can rewrite the two equations in the matrix form:

$$\underset{\mathbf{x}_{t+1}}{\begin{bmatrix} x_{t+1} \\ v_{t+1} \end{bmatrix}} = \underset{A}{\begin{bmatrix} 1 & \Delta t \\ 0 & 1 - \frac{\eta \Delta t}{m} \end{bmatrix}} \underset{\mathbf{x}_t}{\begin{bmatrix} x_t \\ v_t \end{bmatrix}} + \underset{B}{\begin{bmatrix} 0 \\ \frac{\Delta t}{m} \end{bmatrix}} \underset{\mathbf{u}_t}{u_t}$$

Define the one-step quadratic cost function

similar to R(s,a) for MDP

$$\mathbf{x}_t^T Q \mathbf{x}_t + \mathbf{u}_t^T R \mathbf{u}_t \quad \text{for } t = 0, 1, \ldots, N-1$$

state cost

control cost

$$\mathbf{x}_N^T Q_f \mathbf{x}_N$$

final state cost

where Q, Q_f, and R are all symmetric and positive-definite.

Choose $\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{N-1}$ to minimize

$$\sum_{t=0}^{N-1} \left(\mathbf{x}_t^T Q \mathbf{x}_t + \mathbf{u}_t^T R \mathbf{u}_t \right) + \mathbf{x}_N^T Q_f \mathbf{x}_N$$

subject to

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t \quad \text{for } t = 0,1,\ldots,N.$$

and initial state $\mathbf{x}_0$.

To solve for the optimal control, apply dynamic programming.
For $i = 0,1,2,\ldots,N,$ define

$$V_i(\mathbf{x}) = \sum_{t=0}^{N-1} \left(\mathbf{x}_t^T Q \mathbf{x}_t + \mathbf{u}_t^T R \mathbf{u}_t \right) + \mathbf{x}_N^T Q_f \mathbf{x}_N$$

with $\mathbf{x}_i = \mathbf{x}$ and $\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t$ for $t = i, i+1, \ldots, N-1$.
At $i = N$, we have

$$V_N(\mathbf{x}) = \mathbf{x}^T Q_f \mathbf{x}$$

We then work recursively backwards and calculate
$V_N(\mathbf{x}), V_{N-1}(\mathbf{x}), \ldots, V_0(\mathbf{x})$ and $\mathbf{u}_N, \mathbf{u}_{N-1}, \ldots, \mathbf{u}_0$.

The solution is also given recursively:

$$P_N = Q_f$$

For $i = N, N-1, \ldots, 1$,

$$P_{i-1} = Q + A^T P_i A - A^T P_i B (R + B^T P_i B)^{-1} B^T P_i A$$

For $i = 0, 1, 2, \ldots, N-1$,

$$K_i = -(R + B^T P_{i+1} B)^{-1} B^T P_{t+1} A$$

$$\mathbf{u}_i = K_i \mathbf{x}_i$$

The solution applies almost without change to a time-varying system, where the key system parameters A_t and B_t are not constant, but depend on time:

$$\mathbf{x}_{t+1} = A_t \mathbf{x}_t + B_t \mathbf{u}_t \quad \text{for } t = 0, 1, \ldots, N.$$

Motion models. The motion model describes how a (controlled) system evolves over time.

- Discrete-time $x_{t+1} = f(x_t, u_t)$
- Continuous-time $\dot{x} = g(x, u)$

We can obtain a discrete approximation to the continuous-time system by taking a small time-step Δt:

$$\frac{x_{t+1} - x_t}{\Delta t} \approx g(x_t, u_t)$$

$$x_{t+1} \approx x_t + g(x_t, u_t)\Delta t$$

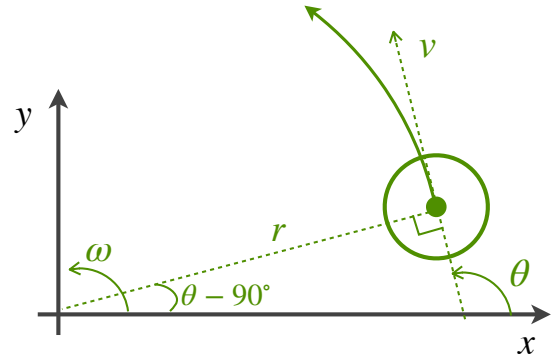
A simple example of discrete-time system dynamics is the linear system model that we have seen:

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t$$

In general, f could be a complex nonlinear function, e.g., a neural network.

Motion models are also called system dynamics models, state-transition models, state-transition equations, … They are all mean the same.

Example. Suppose that a wheeled ground robot is modeled with state $\mathbf{x} = [x, y, \theta]^T$ and is controlled with $\mathbf{u} = [v, \omega]^T$ for linear velocity v and angular velocity ω with respect to a **fixed** coordinate system.



Clearly, $v = \omega r$. At time t, $\mathbf{x}_t = [x, y, \theta]^T$. Then

$$x = r\cos(\theta - 90) = \frac{v}{\omega}\sin(\theta)$$

$$y = r\sin(\theta - 90) = -\frac{v}{\omega}\cos(\theta)$$

After Δt at time $t + 1$, $\mathbf{x}_{t+1} = [x', y', \theta']^T$. Then

$$x' = \frac{v}{\omega}\sin(\theta + \omega\Delta t)$$

$$y' = -\frac{v}{\omega}\cos(\theta + \omega\Delta t)$$

$$\theta' = \theta + \omega\Delta t$$

So

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \frac{v}{\omega}\sin(\theta + \omega\Delta t) - \frac{v}{\omega}\sin\theta \\ -\frac{v}{\omega}\cos(\theta + \omega\Delta t) + \frac{v}{\omega}\cos\theta \\ \omega\Delta t \end{bmatrix}$$

This system dynamic model is nonlinear.

Probabilistic motion models. In reality, the actual robot velocities are subject to noise and different from the commanded ones:

$$\tilde{v} = v + \delta v$$
$$\tilde{\omega} = \omega + \delta\omega$$

where δv and $\delta\omega$ are linear and angular velocity noise following the Gaussian distribution with zero mean and standard deviations that depend on both v and ω. So,

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \frac{\tilde{v}}{\tilde{\omega}}\sin(\theta + \tilde{\omega}\Delta t) - \frac{\tilde{v}}{\tilde{\omega}}\sin\theta \\ -\frac{\tilde{v}}{\tilde{\omega}}\cos(\theta + \tilde{\omega}\Delta t) + \frac{\tilde{v}}{\tilde{\omega}}\cos\theta \\ \tilde{\omega}\Delta t \end{bmatrix}$$

which gives the state-transition equation for a probabilistic system dynamics model similar to the MDP: $T(s, a, s') = p(s' \mid s, a)$.

Model learning. Given data trajectories of the form $(x_0, y_0, \theta_0, v_0, \omega_0, x_1, y_1, \theta_1, v_1, \omega_1, \ldots)$, we can calculate

$$\tilde{v}_t = \frac{\sqrt{(x_{t+1} - x_t)^2 + (y_{t+1} - y_t)^2}}{\Delta t}$$

$$\tilde{\omega}_t = \frac{\theta_{t+1} - \theta_t}{\Delta t}$$

Using v_i and ω_i from the data, we can then estimate the standard deviation of Gaussian noise in v and ω. With all the data, we just need to estimate 2 parameters, the 2 standard deviations for v and ω. This usually works well.

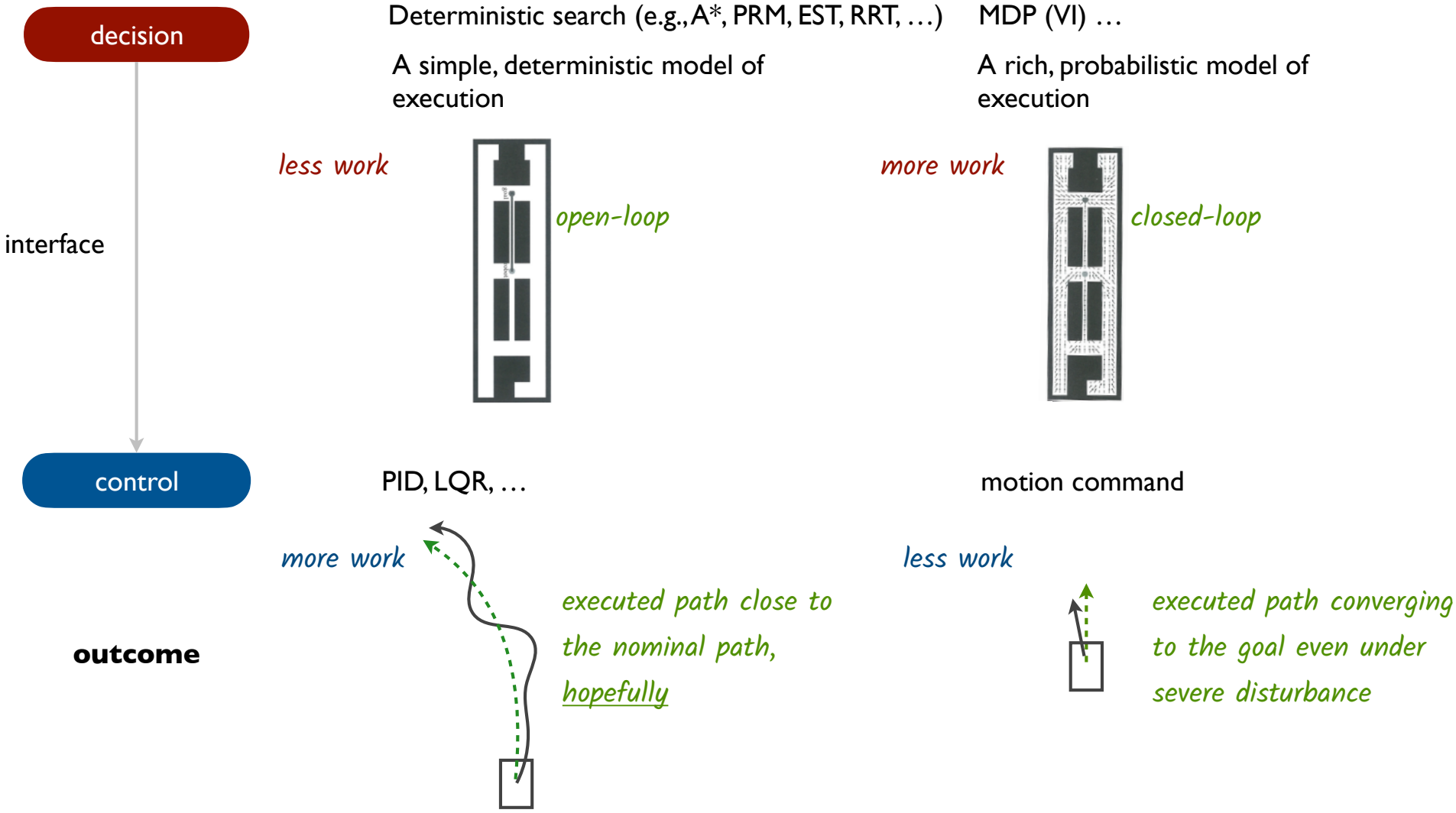
However, the actually noise may not be Gaussian. It may not even easily approximated as Gaussian. For example, it could be multi-modal.

Alternatively, we may try to estimate directly

$$p(x_{t+1}, y_{t+1}, \theta_{t+1} \,|\, x_t, y_t, \theta_t, v_t, \omega_t)$$

as a discrete probability distribution without making the Gaussian noise assumption. This discrete probability function has 8 variables, i.e., 8-dimensional input space. Do we have enough data?

Integrating decision making (planning) and control. Both decision making and control layers together decide on the robot's actions. They form a hierarchical relationship.



- Usually, high-level decision making uses an abstract model and focuses on global choices that are approximately optimal. Low-level control focuses on local choices that require fast response to sometimes unexpected events.
- Decision making requires a reasonable model of motion control and execution.
- Good performance relies on **both** decision making and control to work well together.

Summary.

- A PID controller chooses control actions to drive the error to 0 as fast as possible by considering the current error, cumulative error, and the rate of change in error.
- Tuning the PID controller gain parameters $K_{\mathrm{p}}, K_{\mathrm{i}}$, or K_{d} is critical, but challenging. Various heuristic methods can help.
- LQR provides the optimal control if the system is linear and the cost is quadratic.
- PID is general and does not make any assumptions about the underlying system. LQR makes stronger assumptions and provides stronger performance guarantee, in particular, optimality.
- A rich probabilistic motion model improves the performance of decision making. However, acquiring the model is challenging.
  - Handcrafting the model requires domain knowledge and could be error-prone.
  - Learning the model requires sufficient data.

Required readings.

- [Siegwart, Nourbakhsh & Scaramuzza] Sect 3.1, 3.2.1, 3.2.2, 3.6

Supplementary readings.

- [Siegwart, Nourbakhsh & Scaramuzza] Chap 2

Key concepts.

- Deterministic motion model $x_{t+1} = f(x_t, u_t)$
- Probabilistic motion model $p(x_{t+1} \mid x_t, u_t)$
- PID
- LQR
- Integrating decision making and control