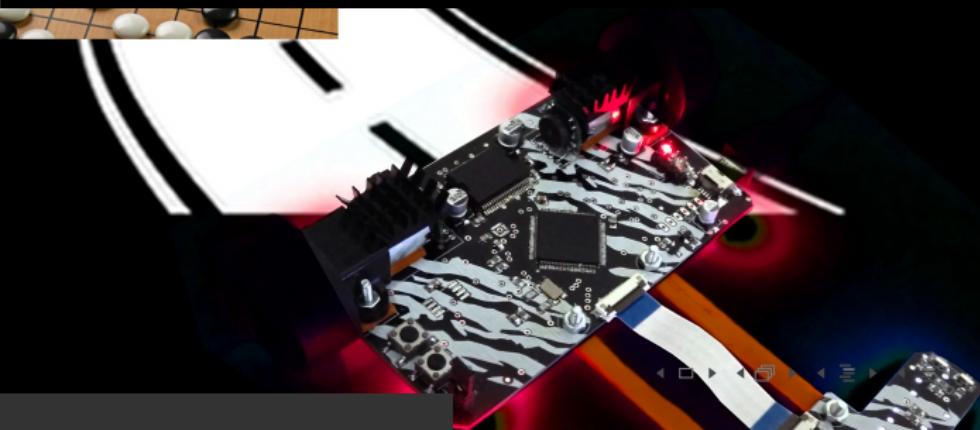


$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \lambda \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

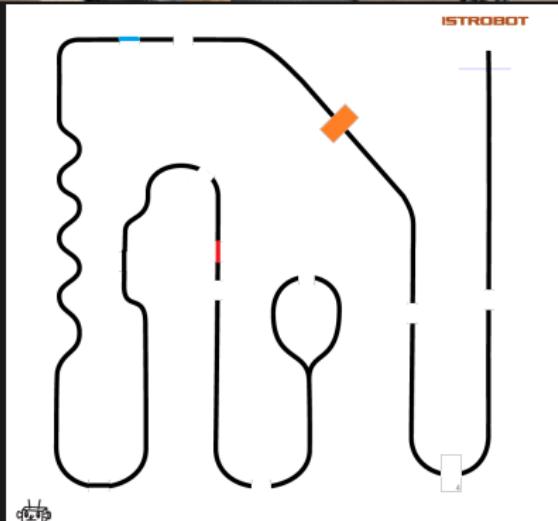
(The New Action Value = The Old Value) + The Learning Rate  $\times$  (The New Information — the Old Information)



- **math**  
- **hardware**  
- **algorithms**  
**for line following robot**  
**Michal CHOVANEC, PhD.**



# line following robot



- fastest line following
- 3 rounds
- obstacles : broken line, brick, loop, colored line

# Istrobot 2024

1. H [1:00,9] [0:52,0] **[0:40,0]** Robot **motoko ice dragon** (Michal Chovanec) Žilina
2. H [1:46,4] **[1:31,8]** [1:38,0] Robot **Pinokio 6** (Peter Hvizdoš) Olcnava 🇸🇰
3. H [::,-,r] [::,-,r] **[2:55,0]** Robot **Pinky Pie** (Katrka Šandová, Hanka Šandová) Polička 🇨🇿

- H [::,-,r] [::,-,r] [::,-,r] Robot **Katarína <3** (Tomáš Volna) Martin 🇸🇰
- H [::,-,r] [::,-,r] [::,-,r] Robot **RSA** (Martin Mišenko, Tadeáš Hudák, Tadeáš Durkáč) Prešov
- H [::,-,r] [::,-,r] [::,-,r] Robot **Rychlosť nenej cesta** (Denis Množil) Křelov 🇸🇰 🇮🇪
- H [::,-,r] [::,-,r] [::,-,r] Robot **HASAK** (Viktor Sova) Uničov 🇨🇿
- H [::,-,r] [::,-,r] [::,-,r] Robot **NPCBOT 2** (Jakub Baluch, Tatiana Glončáková) Kvačany 🇸🇰
- H [::,-,r] [::,-,r] [::,-,r] Robot **Lopata** (Andrea Kuňáková) Pezinok 🇸🇰 🇩🇪
- H [::,-,r] [::,-,r] [::,-,r] Robot **Maskovaný** (Peter Hvizdoš, Andrej Hvizdoš) Olcnava 🇸🇰
- H [::,-,r] [::,-,r] [::,-,r] Robot **XLC Iname** (Adam Ducký, Martin Jaško) Topoľčany 🇸🇰
- H [::,-,r] [::,-,r] [::,-,r] Robot **FRIXbot v.1** (Šimon Motyka) Vrútky 🇸🇰 🍑
- H [::,-,r] [::,-,r] [::,-,r] Robot **SkołTo** (Radosław Děrdá) Olomouc 🇸🇰 🇵🇱
- H [::,-,r] [::,-,r] [::,-,r] Robot **Džufan** (Marek Ratzenböck) Nové Mesto nad Váhom
- H [::,-,r] [::,-,r] [::,-,r] Robot **XLC Tofilius** (Oliver Arzt) Topoľčany 🇸🇰

- - [::,-,r] [::,-,r] [::,-,r] Robot **BaTo** (Šimon Baťo, Konrád Račko) Nové Zámky 🇸🇰
- - [::,-,r] [::,-,r] [::,-,r] Robot **creemesh** (Matúš Šujan) SPŠE Piešťany
- - [::,-,r] [::,-,r] [::,-,r] Robot **Dexter** (Marek Rapavý) SPŠE Piešťany
- - [::,-,r] [::,-,r] [::,-,r] Robot **Ahoj** (Adam Šiška) SPŠE Piešťany
- - [::,-,r] [::,-,r] [::,-,r] Robot **Bitbot** (Danie Dlossy, Peter Brosz) Rovinka 🇩🇪

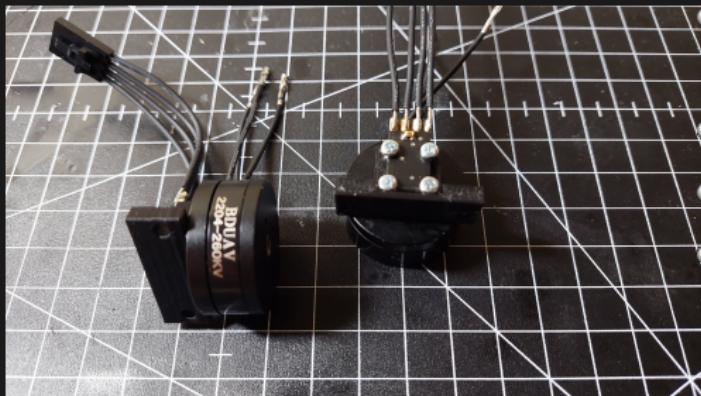
- Robot **KeFI** (Koenrád Račko, Jakub Finda) Nové Zámky 🇸🇰
- Robot **CIPPY** (Matej Vrábel, Alex Mácsadi) Nové Zámky
- Robot **FRIXbot v.2** (Martin Kitko) Diviaky nad Nitricou
- Robot **Volfangus** (Tomáš Nováčik) Topoľčany

- this category registered total 25 robots
- really did participate 21, and 15 of them successfully homologized
- out of them, only 3 did succeed at least in one run
- just 2 robots did successfully all three runs



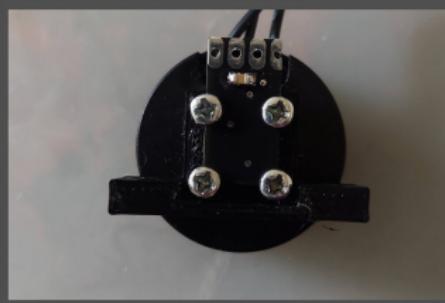
- gear ratio 1:30 (1000 RPM, 0.57kg.cm), item id 2212
- gear ratio 1:50 (590 RPM, 0.86kg.cm), item id 2213
- 1.6A stall current
- magnetic encoder
- **NEVER** buy cheap alternative - looks same, but gears have pure quality, and torque is less than 50% !!!

# brushless motor - 3phase motor

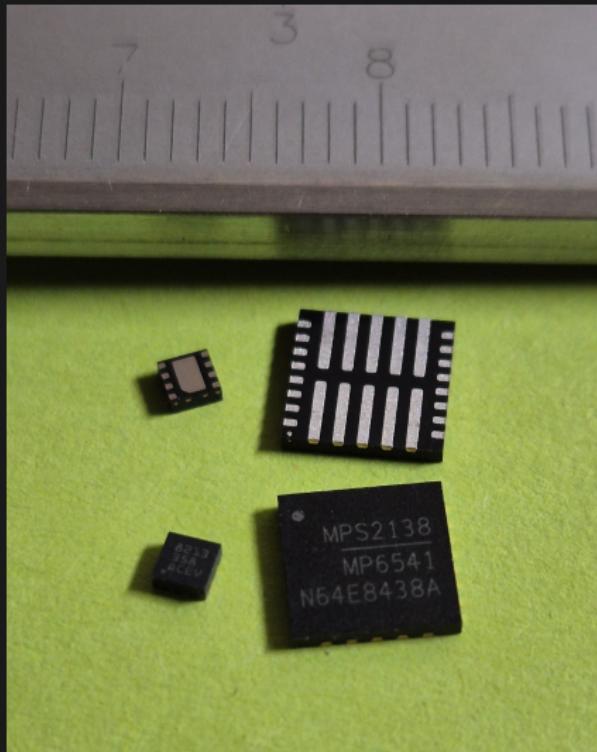


- use "slow" gimbal motors or outrunner for drone
- custom encoder
- custom tire
- extreme huge power - limited by temperature, and core magnetic saturation
- fast response ( $\tau = 10ms$ )

# brushless motor - 3phase motor

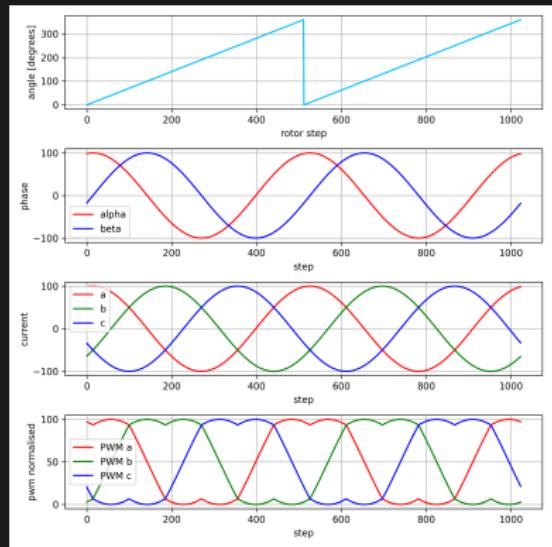
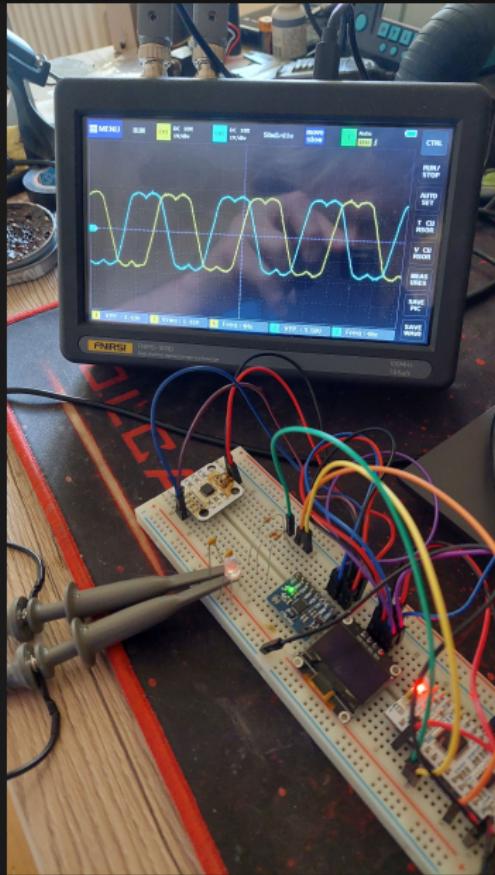


# motor driver

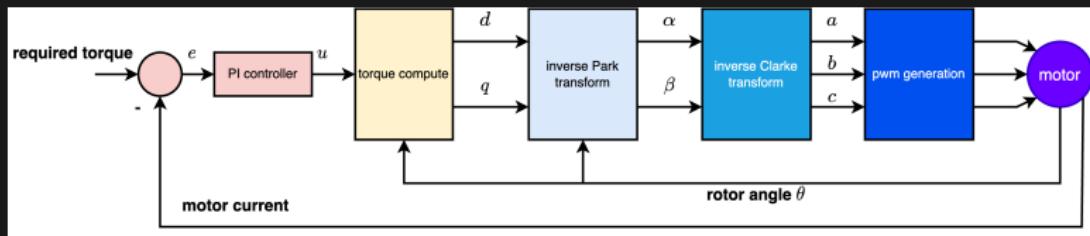


- DC motor
  - DRV8213
  - 4A, 1.65V .. 11V
  - 40W
- Brushless motor
  - MP6541
  - 8A, 4.75V .. 40V
  - 96W
- L293, L298 - obsolete 19th century devices

# FOC - field oriented control



# FOC - field oriented control



follow my tutorial

[https://github.com/michalnand/brushless\\_motor\\_control](https://github.com/michalnand/brushless_motor_control)

# FOC - field oriented control

```
#define SQRT3      ((int32_t)1773)      // sqrt(3)      = 1773/1024
#define SQRT3INV    ((int32_t)591)        // 1/sqrt(3)   = 591/1024

void Motor::set_park(int32_t d, int32_t q, uint32_t theta)
{
    //inverse Park transform
    int32_t alpha = (d*cos_tab(theta) - q*sin_tab(theta))/SINE_TABLE_MAX;
    int32_t beta  = (d*sin_tab(theta) + q*cos_tab(theta))/SINE_TABLE_MAX;

    this->set_clarke(alpha, beta);
}

void Motor::set_clarke(int32_t alpha, int32_t beta)
{
    //inverse Clarke transform
    int32_t a = alpha;
    int32_t b = -(alpha/2) + (SQRT3*beta)/(2*1024);
    int32_t c = -(alpha/2) - (SQRT3*beta)/(2*1024);

    this->set_phases(a, b, c);
}

void Motor::set_phases(int32_t a, int32_t b, int32_t c)
{
    //transform into space-vector modulation, to achieve full voltage range
    int32_t min_val = min3(a, b, c);
    int32_t max_val = max3(a, b, c);

    int32_t com_val = (min_val + max_val)/2;

    //normalise into 0..MOTOR_CONTROL_MAX
    int32_t a_pwm = ((a - com_val)*SQRT3INV)/1024 + MOTOR_CONTROL_MAX/2;
    int32_t b_pwm = ((b - com_val)*SQRT3INV)/1024 + MOTOR_CONTROL_MAX/2;
    int32_t c_pwm = ((c - com_val)*SQRT3INV)/1024 + MOTOR_CONTROL_MAX/2;

    a_pwm = clamp((a_pwm*PWM_PERIOD)/MOTOR_CONTROL_MAX, 0, PWM_PERIOD);
    b_pwm = clamp((b_pwm*PWM_PERIOD)/MOTOR_CONTROL_MAX, 0, PWM_PERIOD);
    c_pwm = clamp((c_pwm*PWM_PERIOD)/MOTOR_CONTROL_MAX, 0, PWM_PERIOD);

    set_pwm(a_pwm, b_pwm, c_pwm);
}
```

# making custom silicone tires

<https://www.kaufland.sk/product/482387287/>



# making custom silicone tires

<https://www.kaufland.sk/product/482387287/>



# making custom silicone tires

<https://www.kaufland.sk/product/482387287/>



# sensors

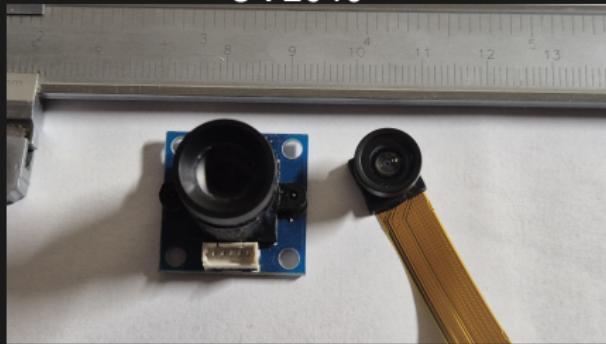
- line sensors
- brick detection
- encoders
- IMU

# line sensors

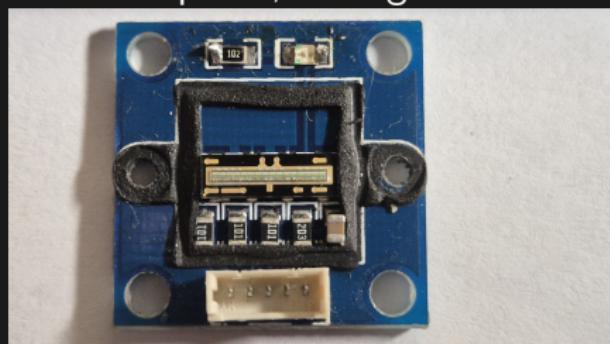
TODO

# camera

any DCMI smd camera, e.g.  
OV2640



line scan camera, TSL1401CL,  
128pixels, analog out

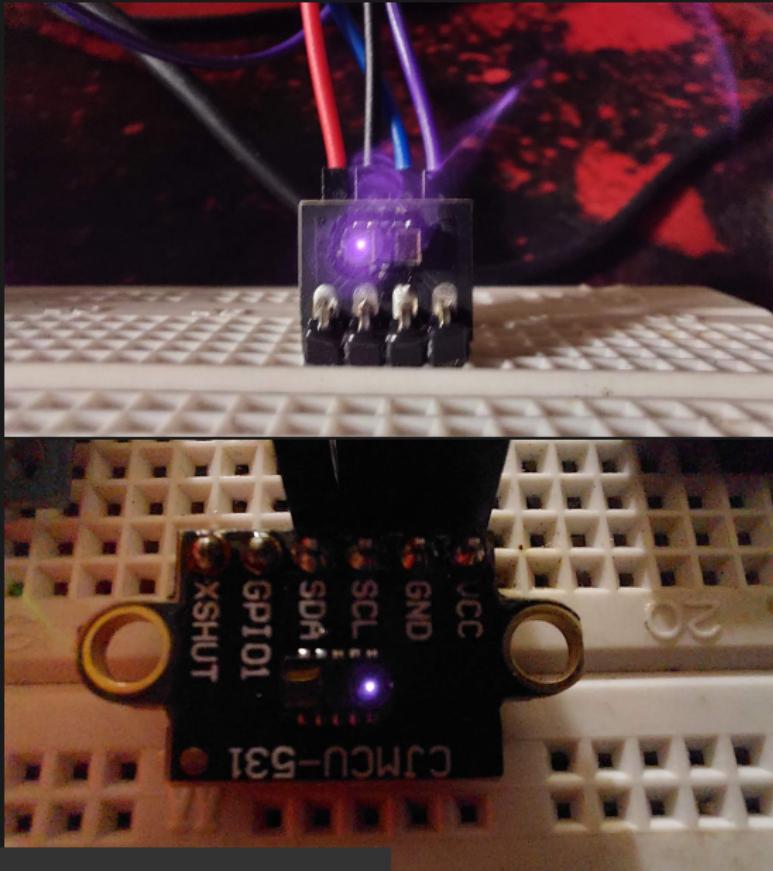


# IR leds sensors

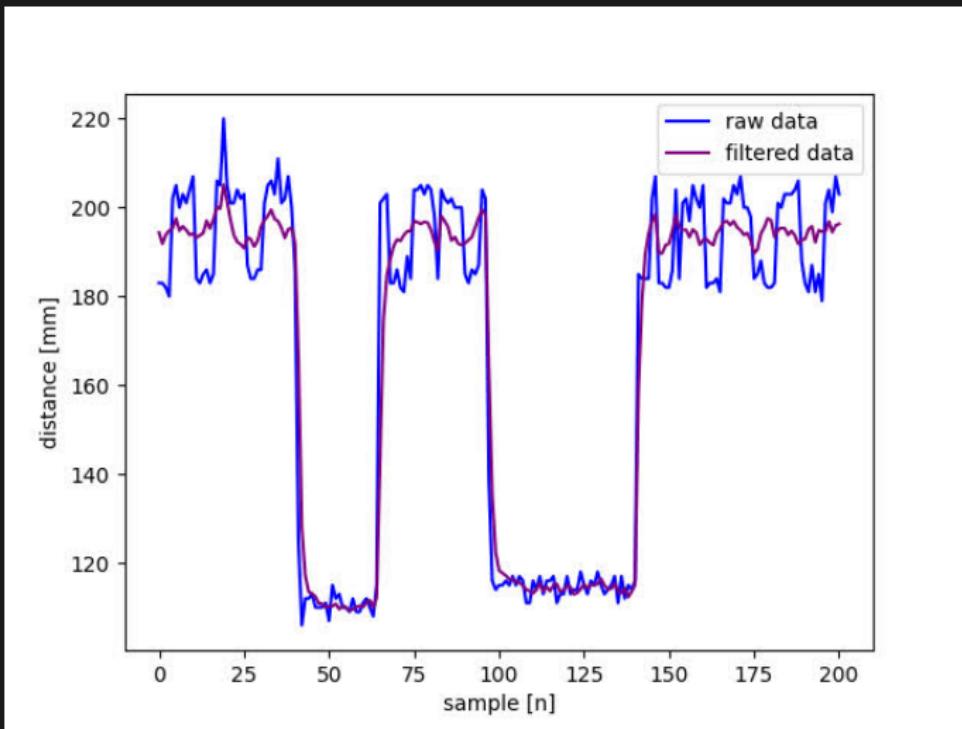
TODO

# laser TOF

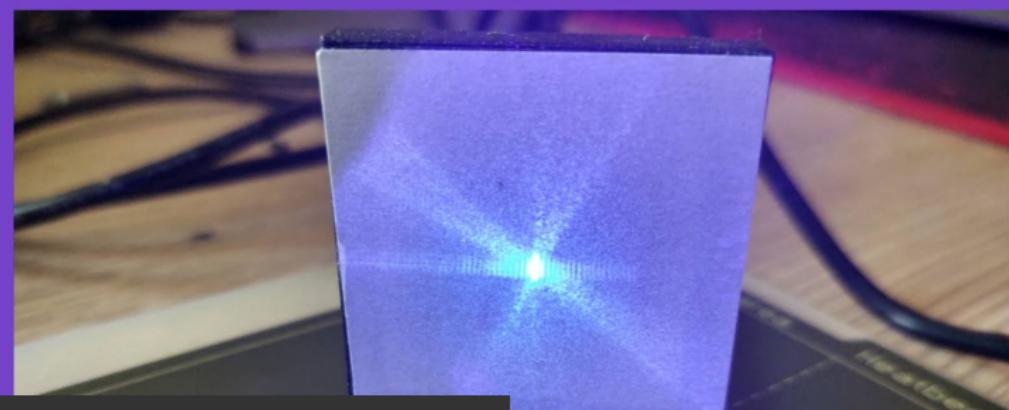
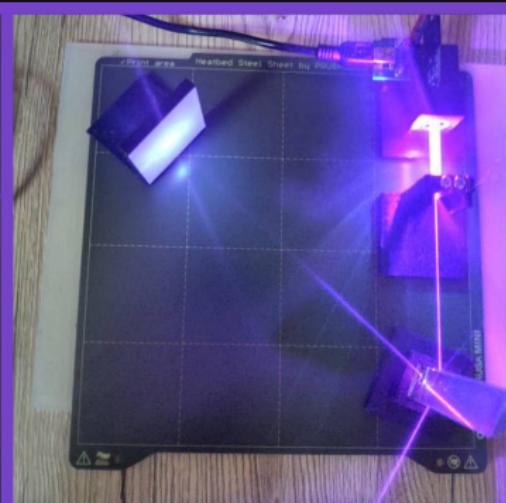
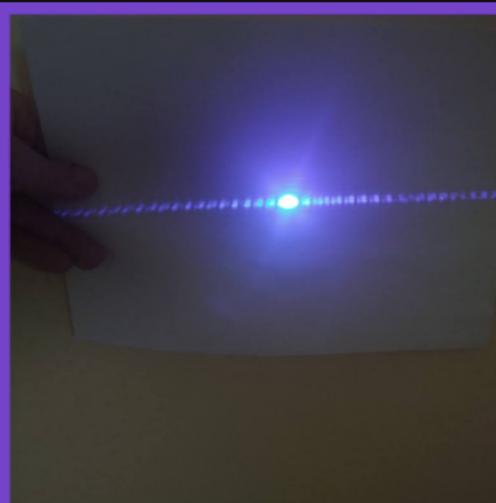
NEVER buy cheep alternative



# laser TOF - filter data



# laser



# encoders

TODO

# robot controll

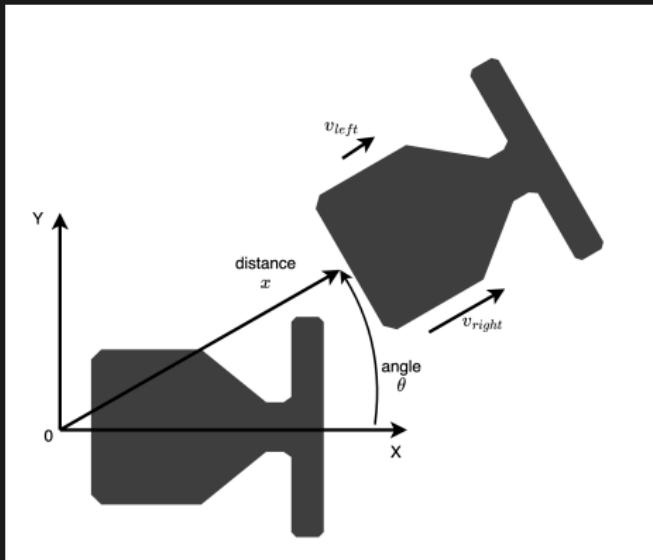
control levels :

- commutation loop : 4kHz, fixed point arithmetics
- velocity loop : 1..4kHz, PI or simple LQR controller
- position loop : 100..500Hz, LQR or MPC controller
- planning loop : 10..20Hz, MPC or neural network

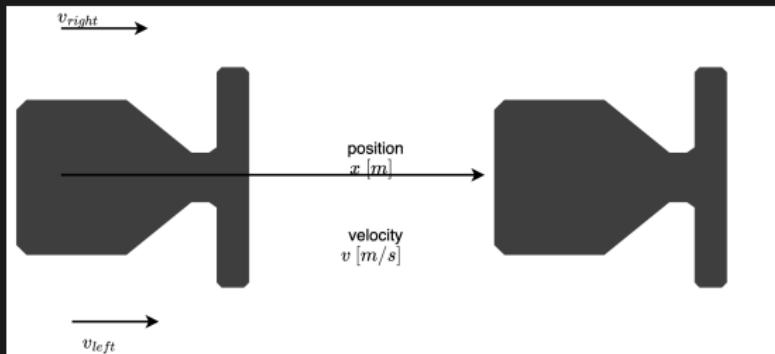
building blocks :

- if else : never captures dynamics
- PID : only SISO systems, only hand tuning
- LQR+Kalman filter : MIMO systems, exact solution
- MPC : MIMO systems, tracking (planning trajectory)
- neural network :RNN (LSTM or GRU), reinforcement learning

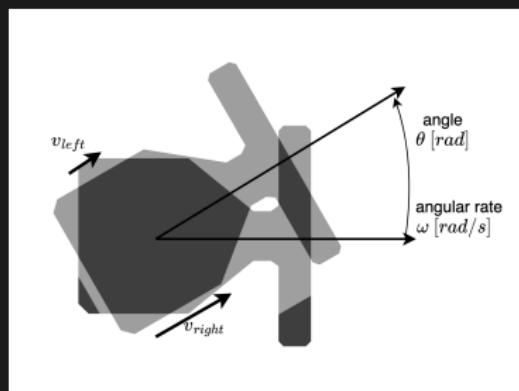
# robot controll - overview



# robot controll - translation



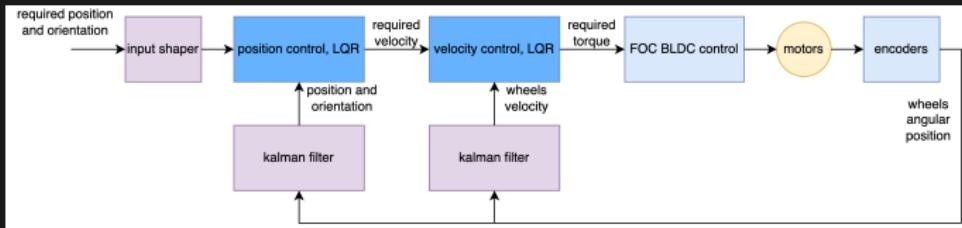
# robot controll - rotation



# robot controll - overview

- position control : distance  $x$  and angle  $\theta$
- required values :  $x_r, \theta_r$
- observed values :  $x, \theta$
- forward :  $x_r(n) = x(n) + speed_r$
- turning :  $\theta_r(n) = \theta(n) + angular\_vel_r$
- full state vector :  $(x, \theta, v, \omega)$
- note 1, : vectors are matrices, with shape  $N \times 1$
- note 2, : nice to have model of dynamics, **planning**

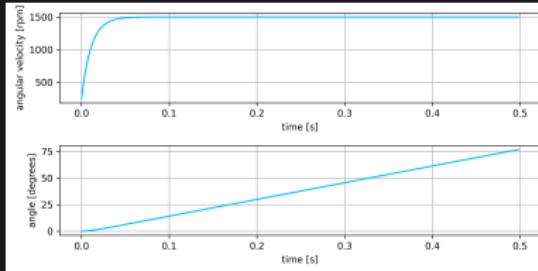
# controller loops



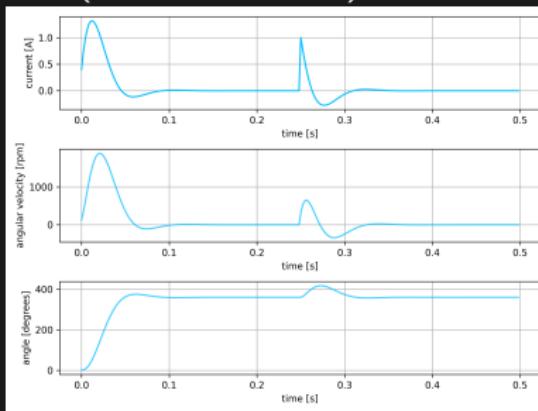
- if we dont know system dynamics (only guessing) :
  - **if else** controller
  - hand tuned **PID** controller
- from known dynamics :
  - Kalman filter
  - LQR, LQG,  $H\infty$
  - **MPC** - planning

# toy example - servo controll

open loop response :



closed loop response (LQR controller) :



# servo state space model

1st order motor  
velocity model

$$\frac{d\omega(t)}{dt} = a\omega(t) + bu(t)$$

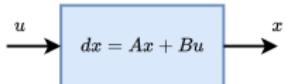
wheel angle

$$\frac{d\theta(t)}{dt} = \omega(t)$$

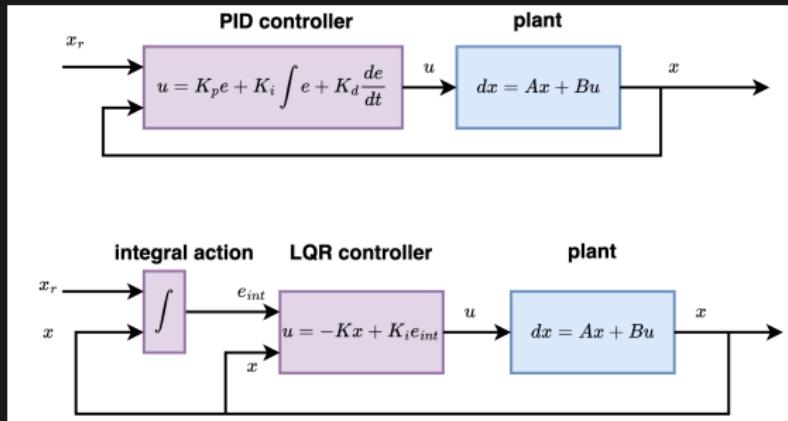
state space model

$$\begin{pmatrix} dx_0 \\ dx_1 \end{pmatrix} = \begin{pmatrix} a & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} + \begin{pmatrix} b \\ 0 \end{pmatrix} (u_0)$$

$$\frac{dx}{dt} = Ax(t) + Bu(t)$$



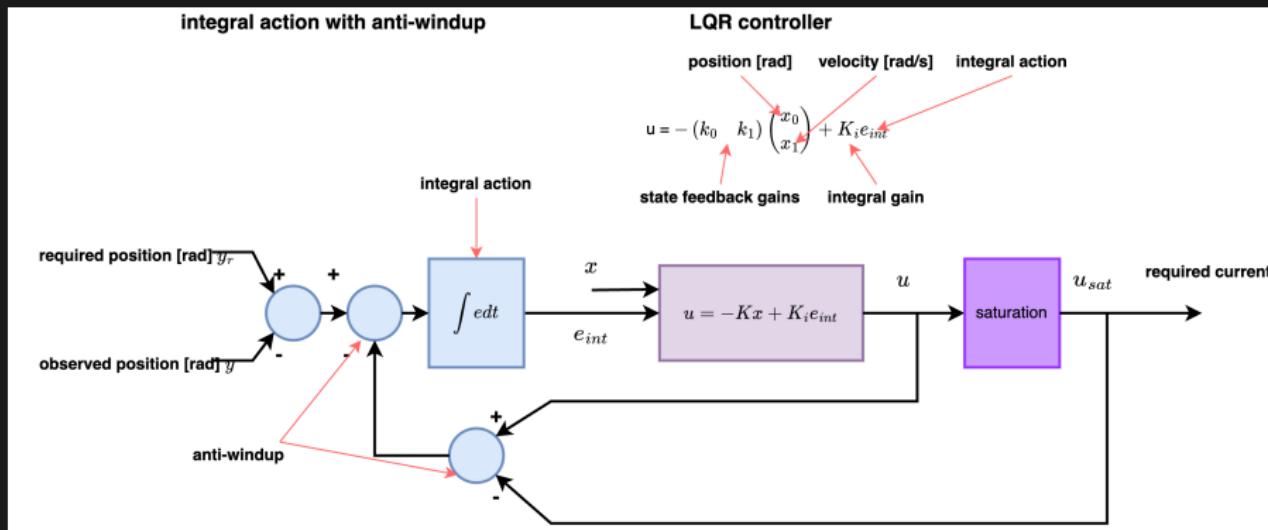
# feedback loop



- **PID** : three **guessed** parameters,  $K_p, K_i, K_d$
- **LQR** : two **computed** matrices,  $K, K_i$

note : continuous modeling (and controllers) have slow inference time - requires Runge-Kutta solvers, always use discrete models and controllers

# complete feedback loop



- input : position  $x_0$ , velocity  $x_1$ , required position  $x_r$
- output : current  $u$
- integral action :  $e_{int}(t) = \int_0^t x_r - x_0 d\tau$
- LQR :  $u = -Kx + K_i e_{int}$

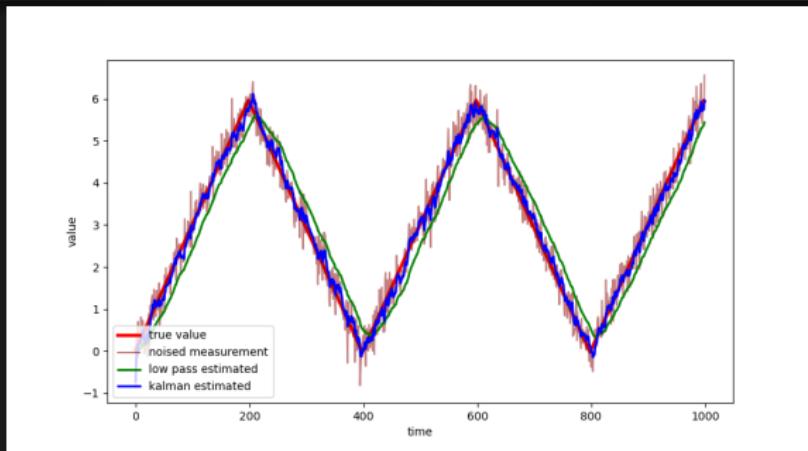
# discrete linear state space model

$$\begin{matrix} 1 \\ \vdots \\ n \end{matrix} \begin{matrix} x(n+1) \\ \vdots \\ n \end{matrix} = \begin{matrix} n \\ \vdots \\ n \end{matrix} A \begin{matrix} 1 \\ \vdots \\ n \end{matrix} \begin{matrix} x(n) \\ \vdots \\ n \end{matrix} + \begin{matrix} m \\ \vdots \\ n \end{matrix} B \begin{matrix} 1 \\ \vdots \\ m \end{matrix} u(n)$$

$$x(n+1) = Ax(n) + Bu(n)$$

- $n$  : system order,  $m$  : system inputs count
- $x(n)$  - system state, matrix  $n \times 1$
- $u(n)$  - control input, matrix  $m \times 1$
- $A$  - transient matrix,  $n \times n$
- $B$  - input matrix,  $n \times m$

# toy example - kalman filter object tracking



low pass filter (exponential moving average)

$$\hat{x}(n+1) = \alpha \hat{x}(n) + (1 - \alpha)x(n)$$

kalman filter

$$\hat{x}(n+1) = A\hat{x}(n) + K(x(n) - \hat{x}(n))$$

$$\hat{x}(n+1) = A\hat{x}(n) + Bu(n) + K(x(n) - \hat{x}(n))$$

# toy example - kalman filter object tracking

constant position model

$$(x)_{n+1} = (1) (x)_n$$

constant velocity model

$$\begin{pmatrix} v \\ x \end{pmatrix}_{n+1} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} v \\ x \end{pmatrix}_n$$

constant acceleration model

$$\begin{pmatrix} a \\ v \\ x \end{pmatrix}_{n+1} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} a \\ v \\ x \end{pmatrix}_n$$

# red pill section

# LQR - optimal control

controller

$$u(n) = -Kx(n)$$

loss (cost) function is quadratic, with weighting terms  $Q$  and  $R$

$$\mathcal{L} = \sum_n^\infty x^T(n)Qx(n) + u^T(n)Ru(n)$$

$$s.t. \quad x(n+1) = Ax(n) + Bu(n)$$

matrix  $K$  is obtained by solving Riccati ARE

$$P(n-1) = Q + A^T P(n)A - A^T P(n)B(B^T P(n) + R)^{-1}B^T P(n)A$$

where  $P(\infty) = Q$

$$K = R^{-1}B^T P(0)$$

# system identification

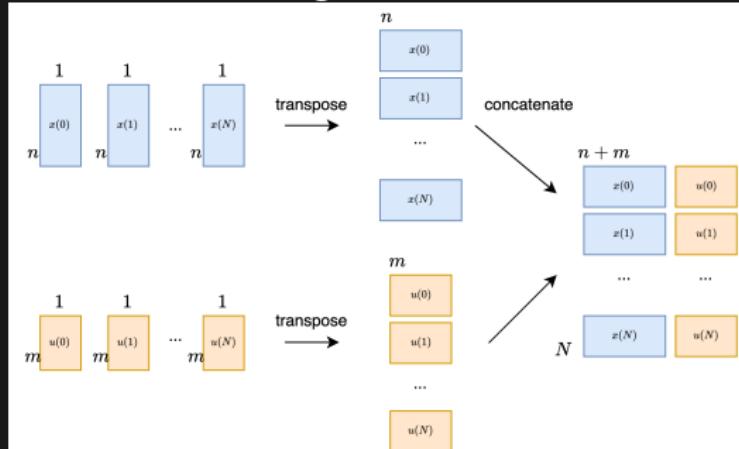
- generate input sequence :  $u$
- measure data :  $x$
- fitting model  $\hat{A}$ ,  $\hat{B}$  into data
- predicted data :  $\hat{x}$

$$\mathcal{L} = \sum_{n=0}^N (x(n) - \hat{x}(n))^2$$

$$s.t. \quad \hat{x}(n+1) = \hat{A}x(n) + \hat{B}u(n)$$

# system identification

we can stack  $x$  and  $u$  into single matrix  $\tilde{X}$



same for matrices  $\hat{A}, \hat{B}$  into  $\tilde{M}$ ,  
also for  $\hat{x} : \hat{X}$ , to get compact form

$$\hat{X} = \tilde{M}\tilde{X}$$

# system identification - least squares

$$\hat{X} = \tilde{M}\tilde{X}$$

**which is the least squares problems**

can be solved by Moore-Penrose matrix pseudoinverse

$\tilde{M} = \text{numpy.linalg.lstsq}(\tilde{X}, X)$

# model predictive control

loss (cost) function is quadratic, with weighting terms  $Q$  and  $R$ ,

$$\mathcal{L} = \sum_h^{H-1} x^T(h) Q x(h) + d u^T(h) R_{\Delta} u(h)$$

$$u(n) = u(n-1) +_{\Delta} u(n)$$

$$s.t. \quad x(n+1) = Ax(n) + Bu(n)$$

where :

- $H$  is prediction horizon steps
- $A$  is matrix,  $n \times n$
- $B$  is matrix,  $n \times m$
- $Q$  is matrix,  $n \times n$
- $R$  is matrix,  $m \times m$
- $\Delta u$  is controller output
- where  $n$  is system orders, and  $m$  system inputs count

# model predictive control

unwrap into form where only  $x(n)$  is used to predict all other  $x$

$$x(n+1) = Ax(n) + Bu(n)$$

$$= Ax(n) + B(u(n-1) + \Delta u(n))$$

$$x(n+2) = Ax(n+1) + B(u(n) + \Delta u(n+1))$$

$$= A^2x(n) + (AB + B)u(n-1) + (AB + B)\Delta u(n) + B\Delta u(u)$$

$$x(n+3) = Ax(n+2) + B(u(n+1) + \Delta u(n+2))$$

$$= A^3x(n) + (A^2 + AB + B)u(n-1)$$

$$+ (AB + B)\Delta u(n+1) + B\Delta u(n+2)$$

...

# model predictive control

rewrite into matrix form

$$\begin{bmatrix} x(n+1) \\ x(n+2) \\ \dots \\ x(n+H) \end{bmatrix} = \begin{bmatrix} A^1 \\ A^2 \\ \dots \\ A^H \end{bmatrix} x(n) + \begin{bmatrix} B \\ AB \\ \dots \\ \sum_{i=0}^{H-1} A^i B \end{bmatrix} u(n-1) + \\ + \begin{bmatrix} B & 0 & \dots & 0 \\ AB+B & B & \dots & 0 \\ \dots & \dots & \dots & \dots \\ \sum_{i=0}^{H-1} A^i B & \sum_{i=0}^{H-2} A^i B & \dots & B \end{bmatrix} \begin{bmatrix} \Delta u(n) \\ \dots \\ \Delta u(n+H-1) \end{bmatrix}$$

in compat form

$$\tilde{X} = \Psi x(n) + \Omega u(n-1) + \Theta \tilde{\Delta} U$$

# model predictive control

quadratic loss (cost) function

$$\mathcal{L} = {}_{\Delta}\tilde{U}^T \tilde{R}_{\Delta} \tilde{U} + (\tilde{X}_r - \tilde{X})^T \tilde{Q} (\tilde{X}_r - \tilde{X})^T$$

after substitution

$$S = \tilde{X}_r - \Psi \tilde{X} - \Omega u(n-1)$$

$$\begin{aligned}\mathcal{L} &= {}_{\Delta}\tilde{U}^T \tilde{R}_{\Delta} \tilde{U} + (S - \Theta_{\Delta} \tilde{U})^T \tilde{Q} (S - \Theta_{\Delta} \tilde{U}) \\ &= {}_{\Delta}\tilde{U}^T \tilde{R}_{\Delta} \tilde{U} + S^T \tilde{Q} S - S^T \tilde{Q} \Theta_{\Delta} \tilde{U} - {}_{\Delta}\tilde{U}^T \Theta^T \tilde{Q} S + {}_{\Delta}\tilde{U}^T \Theta^T \tilde{Q} \Theta_{\Delta} \tilde{U}\end{aligned}$$

# model predictive control

find derivative with respect to  $\Delta \tilde{U}$

$$\frac{\partial \mathcal{L}}{\partial \Delta \tilde{U}} :$$

$$\frac{\partial \Delta \tilde{U}^T \tilde{R}_\Delta \tilde{U}}{\partial \Delta \tilde{U}} = 2 \tilde{R}_\Delta \tilde{U}$$

$$\frac{\partial S^T \tilde{Q} S}{\partial \Delta \tilde{U}} = 0$$

$$\frac{\partial -S^T \tilde{Q} \Theta_\Delta \tilde{U}}{\partial \Delta \tilde{U}} = -\Theta^T \tilde{Q} S$$

$$\frac{\partial -\Delta \tilde{U}^T \Theta^T \tilde{Q} S}{\partial \Delta \tilde{U}} = -\Theta^T \tilde{Q} S$$

$$\frac{\partial \Delta \tilde{U}^T \Theta^T \tilde{Q} \Theta_\Delta \tilde{U}}{\partial \Delta \tilde{U}} = 2 \Theta^T \tilde{Q} \Theta_\Delta \tilde{U}$$

# model predictive control

put derivate equal to zero, and solve

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \tilde{U}} &= 2\tilde{R}_\Delta \tilde{U} - 2\Theta^T \tilde{Q}S + 2\Theta^T \tilde{Q}\Theta_\Delta \tilde{U} \\ 0 &= 2\tilde{R}_\Delta \tilde{U} - 2\Theta^T \tilde{Q}S + 2\Theta^T \tilde{Q}\Theta_\Delta \tilde{U} \\ (\tilde{R} + \Theta^T \tilde{Q}\Theta)_\Delta \tilde{U} &= \Theta^T \tilde{Q}S\end{aligned}$$

and obtain solution for model predictive control

$$\Delta \tilde{U} = (\tilde{R} + \Theta^T \tilde{Q}\Theta)^{-1} \Theta^T \tilde{Q}S$$

# Q&A



- <https://github.com/michalnand/>
- michal.nand@gmail.com