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

(The New Action Value = The Old Value) + The Learning Rate × (The New Information − the Old Information)

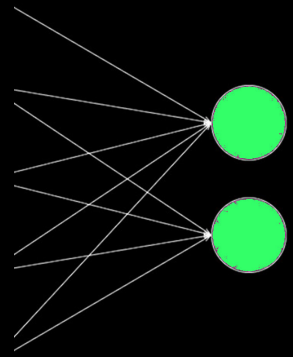
Hidden

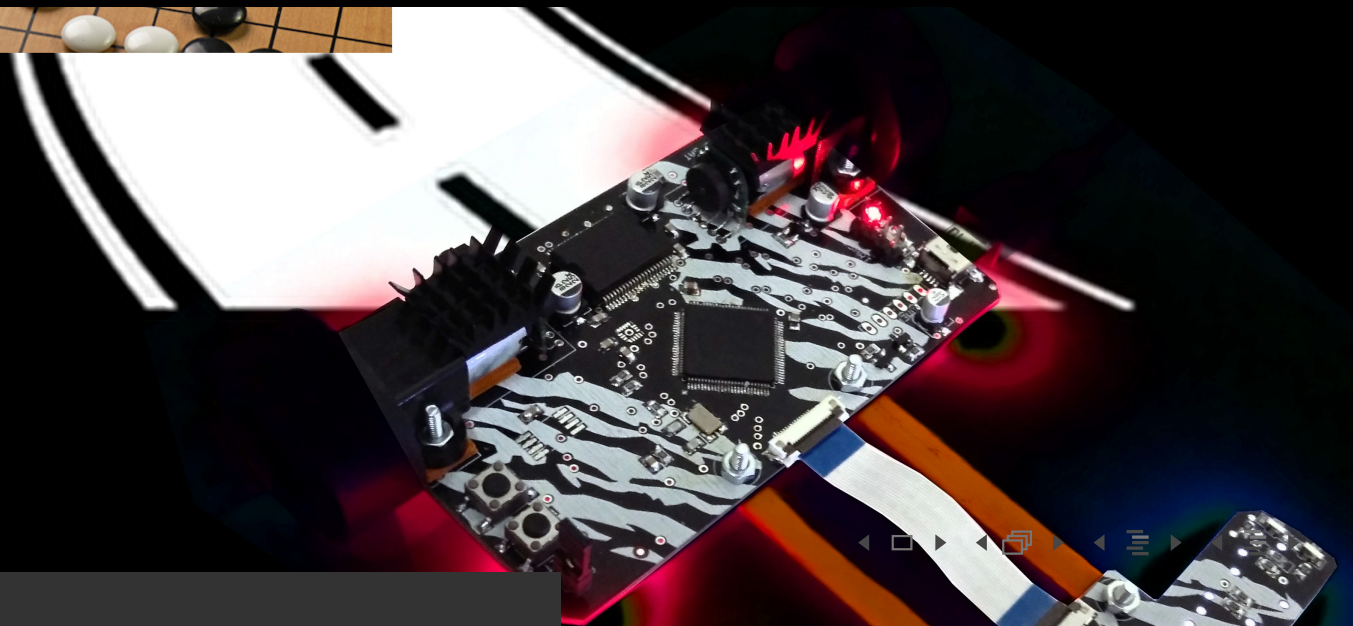
Output

- math
- hardware
- algorithms

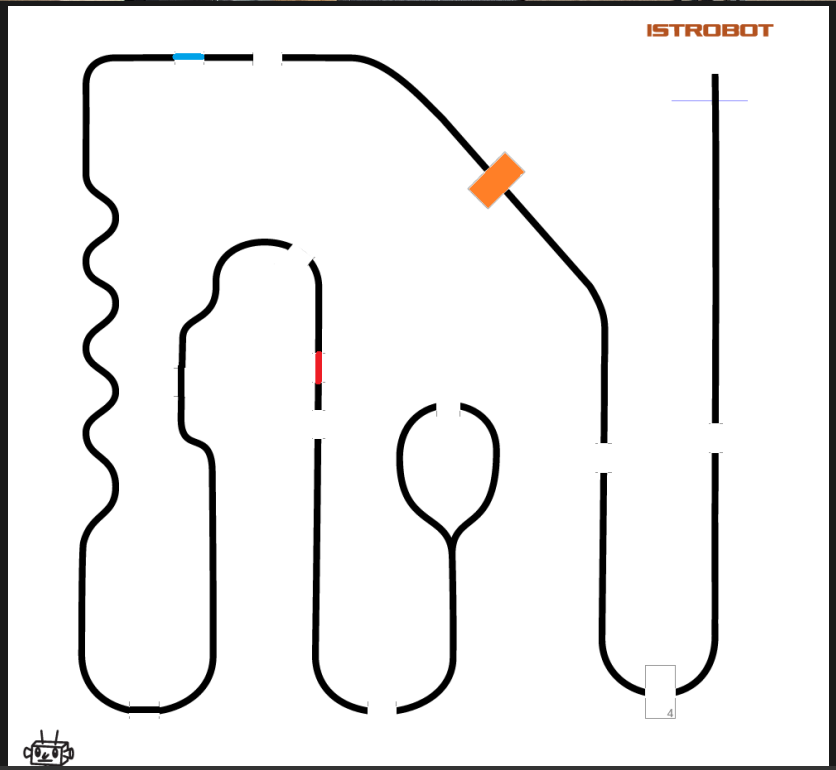
for line following robot

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 [-:--,-] [-:--,-] **[2:55,0]** Robot **Pinky Pie** (Katka Šandová, Hanka Šandová) Polička

---

- H [-:--,-] [-:--,-] [-:--,-] Robot **Katarína <3** (Tomáš Volna) Martin
- H [-:--,-] [-:--,-] [-:--,-] Robot **RSA** (Martin Mišenko, Tadeáš Hudák, Tadeáš Durkáč) Prešov
- H [-:--,-] [-:--,-] [-:--,-] Robot **Rychlost není cesta** (Denis Množil) Křelov
- H [-:--,-] [-:--,-] [-:--,-] Robot **HASAK** (Viktor Sova) Uničov
- H [-:--,-] [-:--,-] [-:--,-] Robot **NPCBOT2** (Jakub Baluch, Tatiana Glončáková) Kvačany
- H [-:--,-] [-:--,-] [-:--,-] Robot **Lopata** (Andrea Kuňáková) Pezinok
- H [-:--,-] [-:--,-] [-:--,-] Robot **Maskovaný** (Peter Hvizdoš, Andrej Hvizdoš) Olcnava
- H [-:--,-] [-:--,-] [-:--,-] Robot **XLC !name** (Adam Ducký, Martin Jaško) Topoľčany
- H [-:--,-] [-:--,-] [-:--,-] Robot **FRIXbot v.1** (Šimon Motyka) Vrútky
- H [-:--,-] [-:--,-] [-:--,-] Robot **SkolTo** (Radim Děrda) Olomouc
- H [-:--,-] [-:--,-] [-:--,-] Robot **Džufan** (Marek Ratzenböck) Nové Mesto nad Váhom
- H [-:--,-] [-:--,-] [-:--,-] Robot **XLC Tofilius** (Oliver Arzt) Topoľčany

---

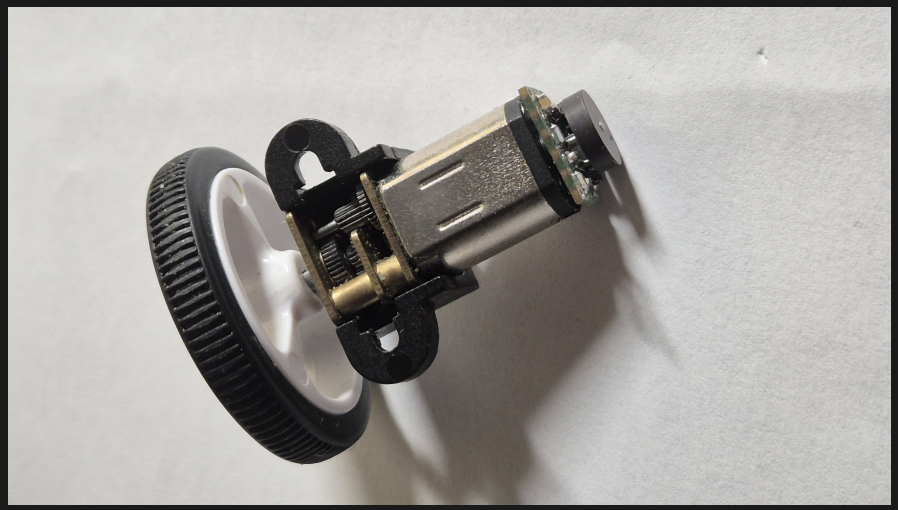
- - [-:--,-] [-:--,-] [-:--,-] Robot **BaTo** (Šimon Baťo, Konrád Račko) Nové Zámky
- - [-:--,-] [-:--,-] [-:--,-] Robot **creemesh** (Matúš Šujan) SPŠE Piešťany
- - [-:--,-] [-:--,-] [-:--,-] Robot **Dexter** (Marek Rapavý) SPŠE Piešťany
- - [-:--,-] [-:--,-] [-:--,-] Robot **Ahoj** (Adam Šiška) SPŠE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

---

- ~~Robot **KoFi** (Konrád Račko, Jakub Finda) Nové Zámky~~
- ~~Robot **CIPPY** (Matej Vrábel, Alex Mácsadi) Nové Zámky~~
- ~~Robot **FRIXbot v.2** (Martin Kitko) Diviaky nad Nitricou~~
- ~~Robot **Volfgangus** (Tomáš Nováčik) Topoľčany~~

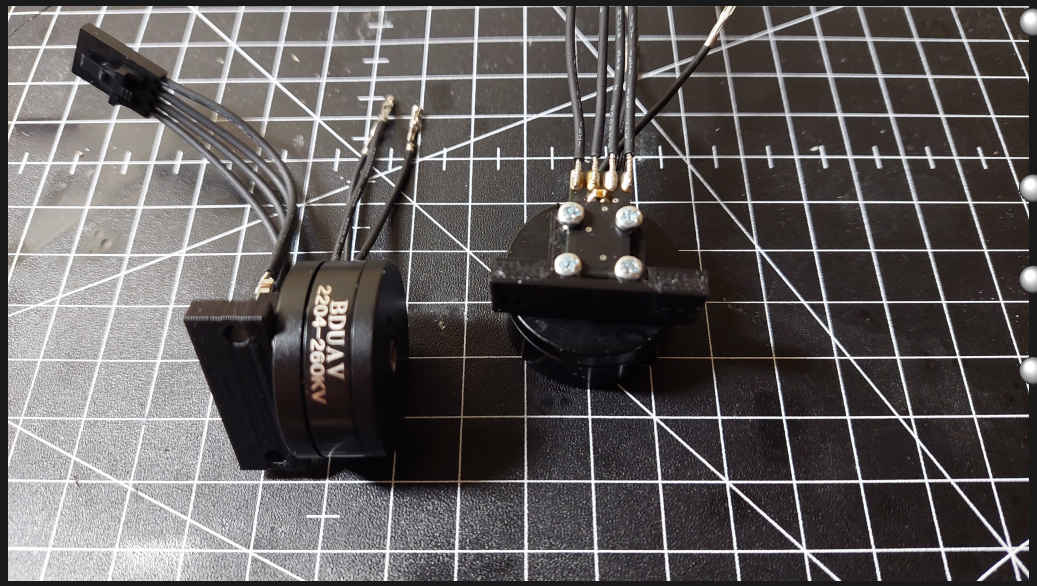
- this category registered total 25 robots
- really did particicpate 21, and 15 of them succesfully homologized
- out of them, only 3 did succeed at least in one run
- just 2 robots did succesfully all three runs

DC motor - Pololu.com



- 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 cheep alternative - looks same, but gears have pure quality, and torque is less than 50% !!!

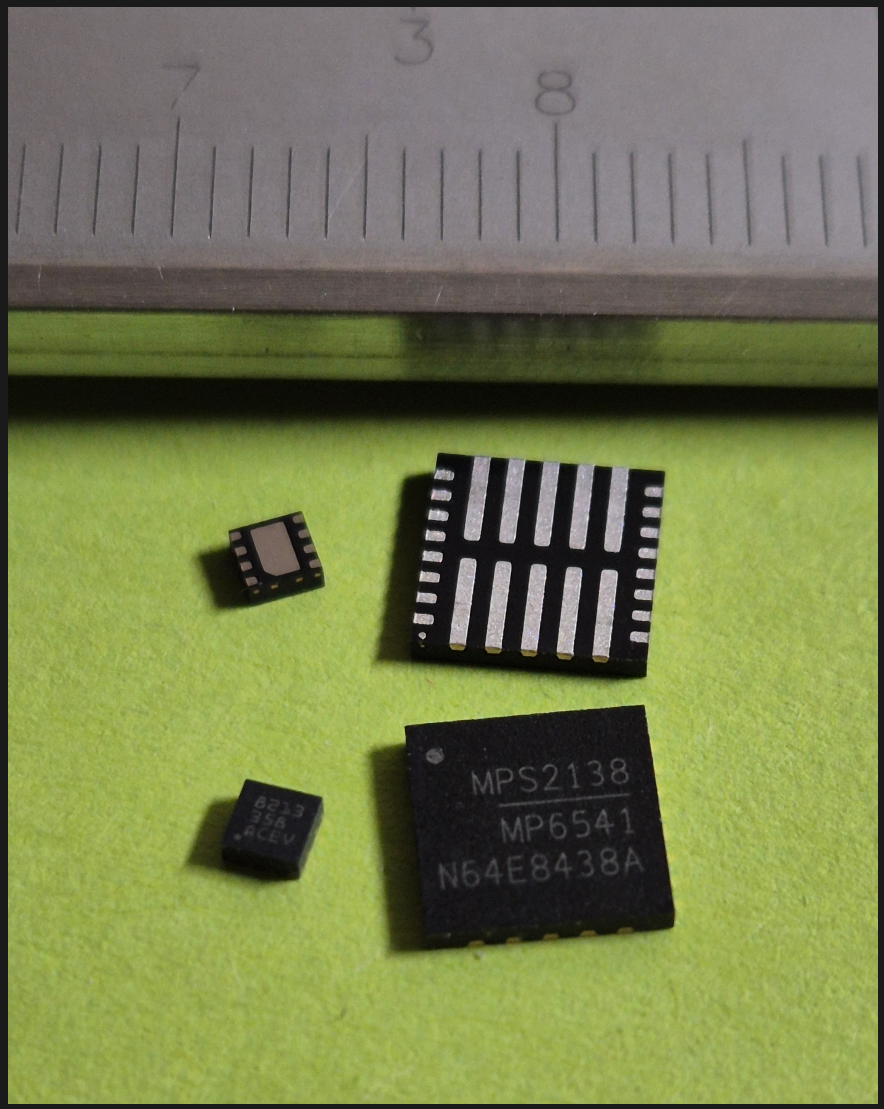
brushless motor - 3phase motor



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

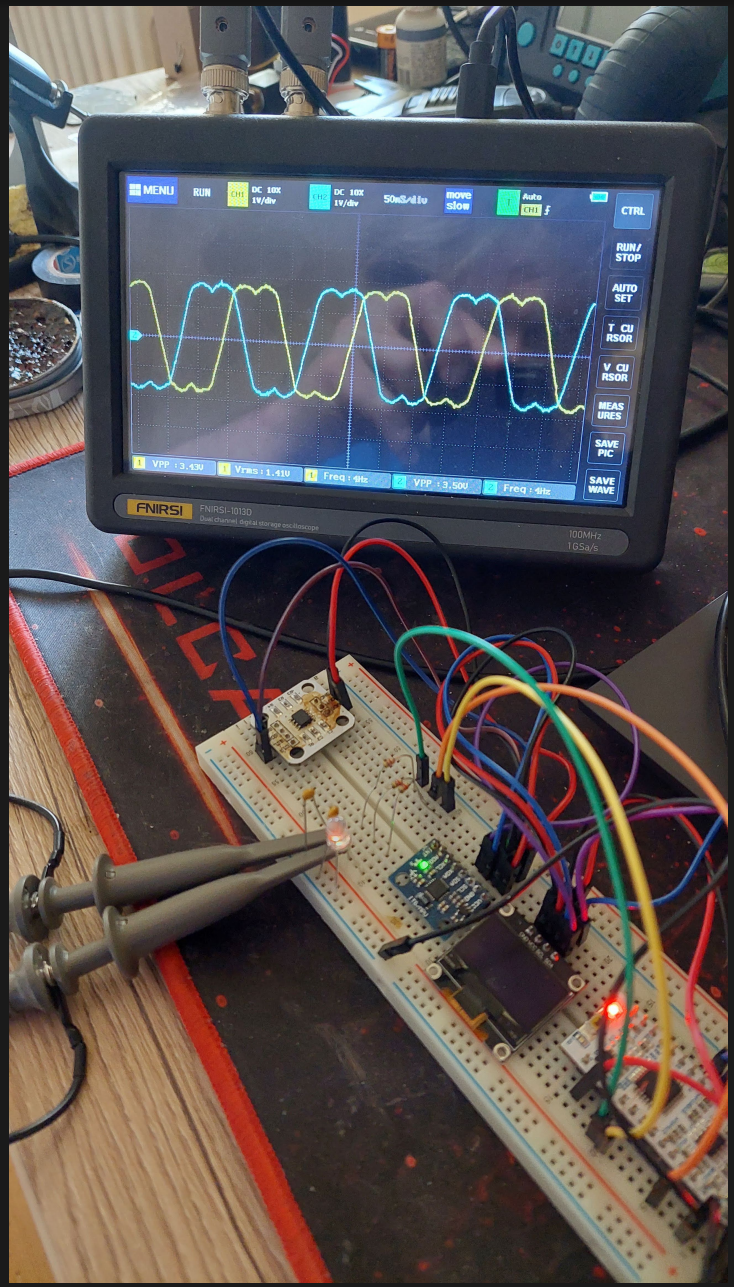
brushless motor - 3phase motor

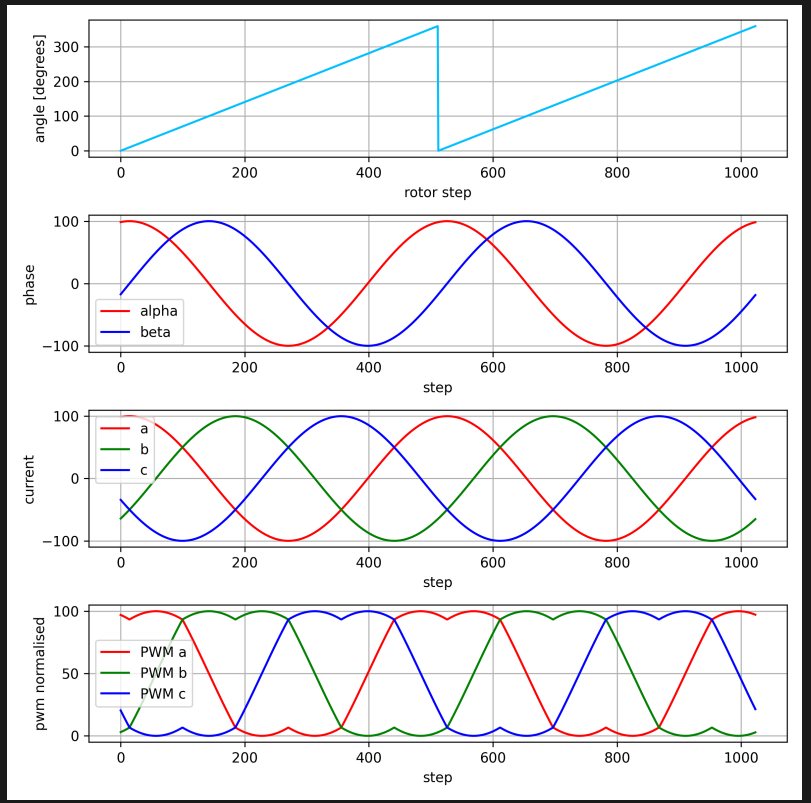
motor driver



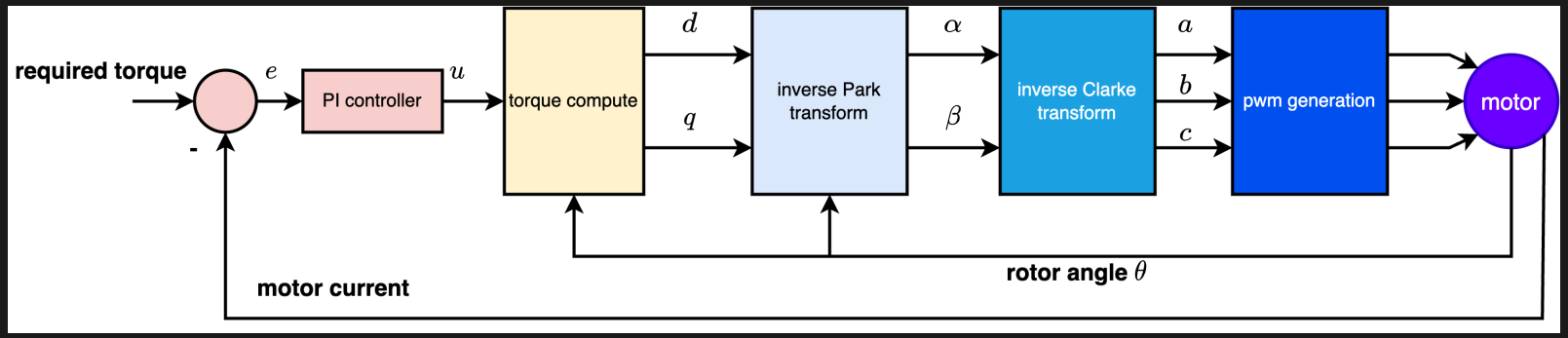
- DC motor
  - DRV8213
  - 4A, 1.65V .. 11V
  - 40W
- Brushless motor
  - MP6541
  - 8A, 4.75V .. 40V
  - 96W
- L293, L298 - obsolete 19th century devices

FOC - field oriented control





FOC - field oriented control



follow my tutorial

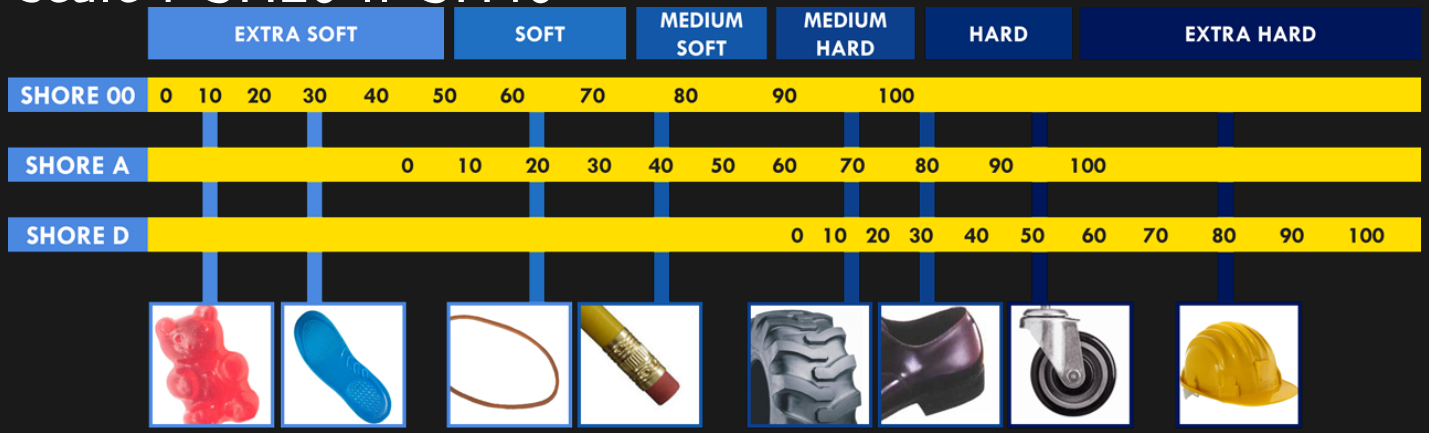
https://github.com/michalnand/brushless_motor_control

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);
}
```

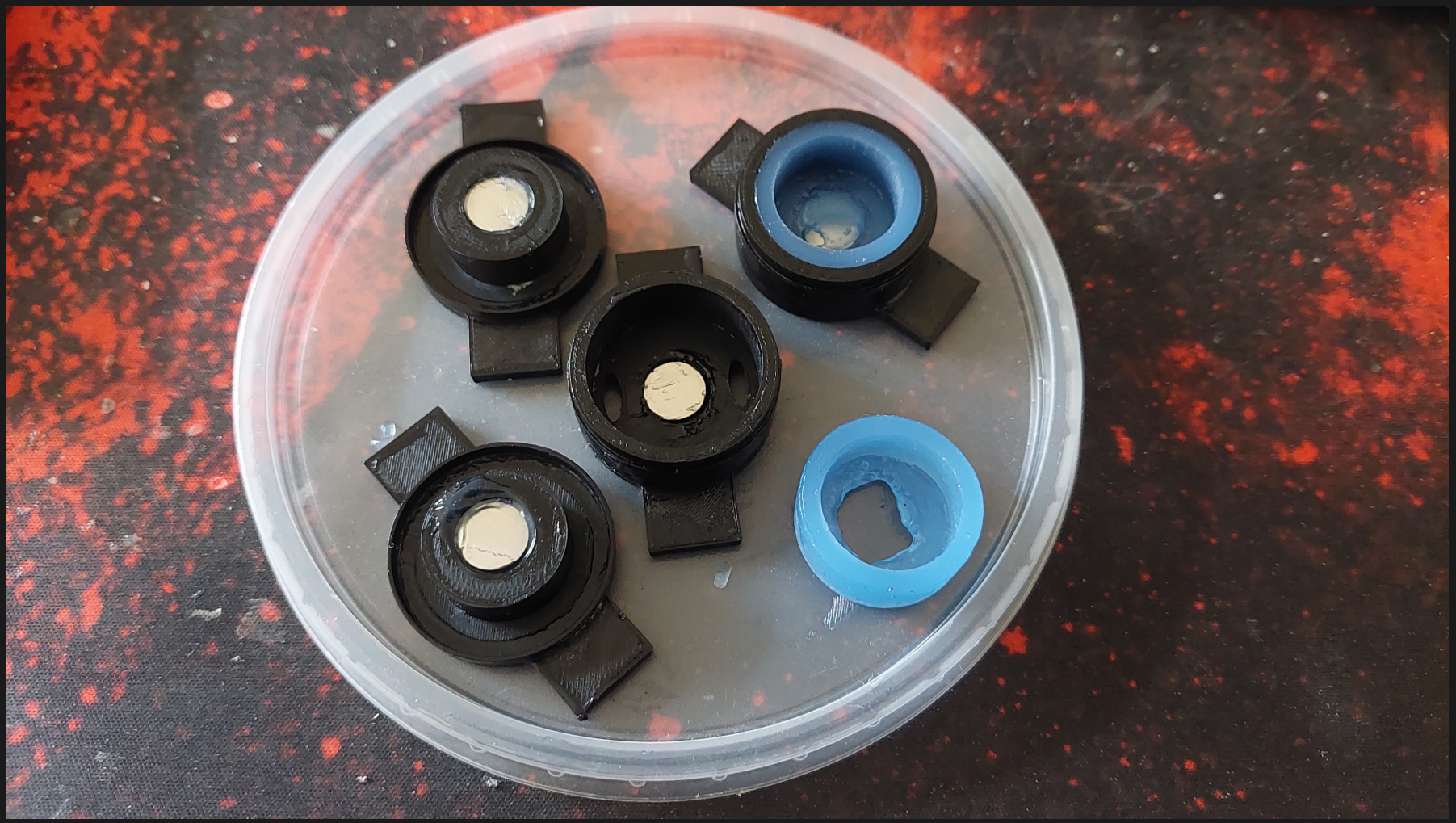
making custom silicone tires

shore scale : SH20 .. SH40



https://www.mojefarby.sk/pour-art-silikon-na-vyrobu-formy-asf-20sh-1kg-2/

making custom silicone tires

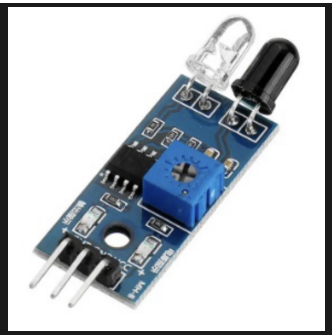


making custom silicone tires

sensors

- line sensors
- brick detection
- encoders
- IMU

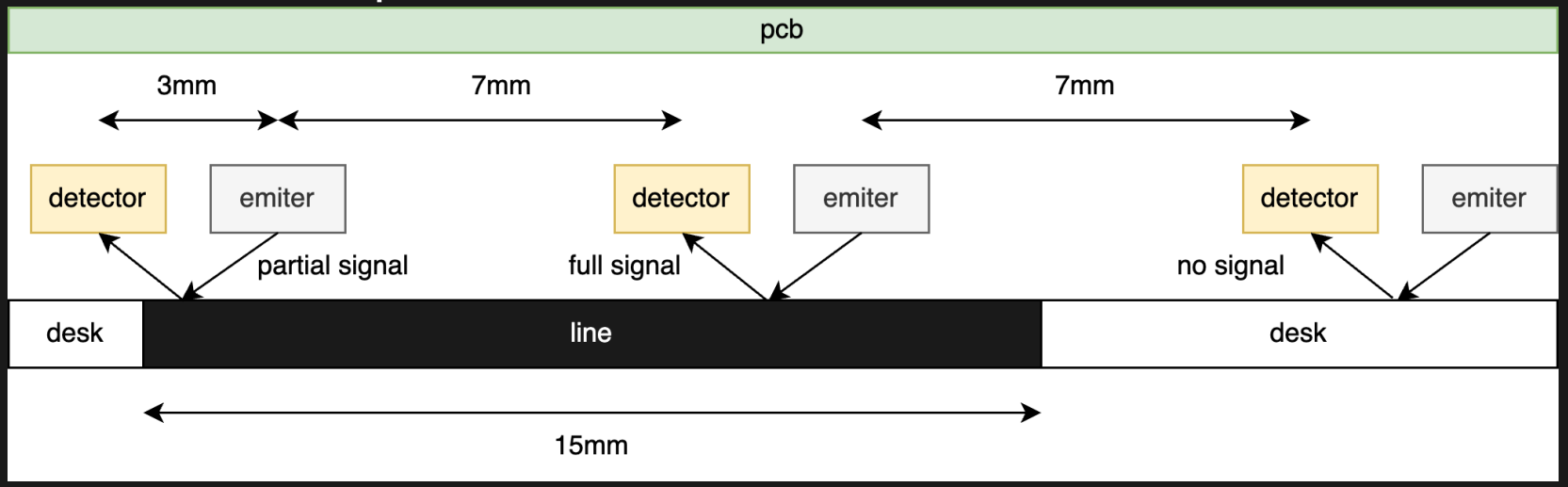
line sensors



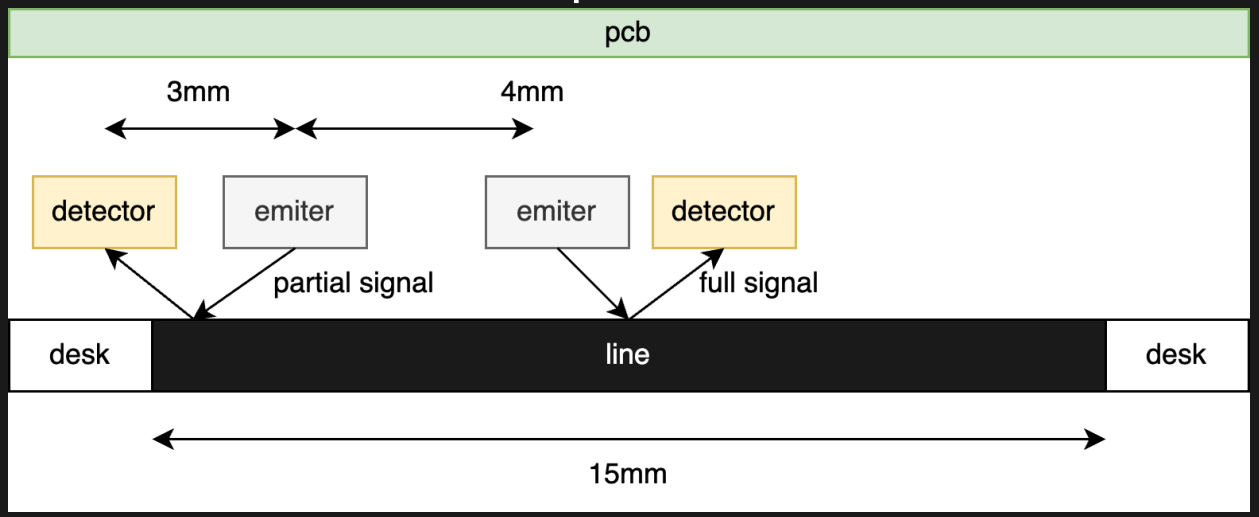
- CNY70 - IR, huge, obsolete
- smd phototransistor + white LED : 550nm can see colored line
- RGB senzor, APDS9950 : can see full color information
- linear or quadratic interpolation : never use only 8 values

line sensors - geometry

correct sensors placement :

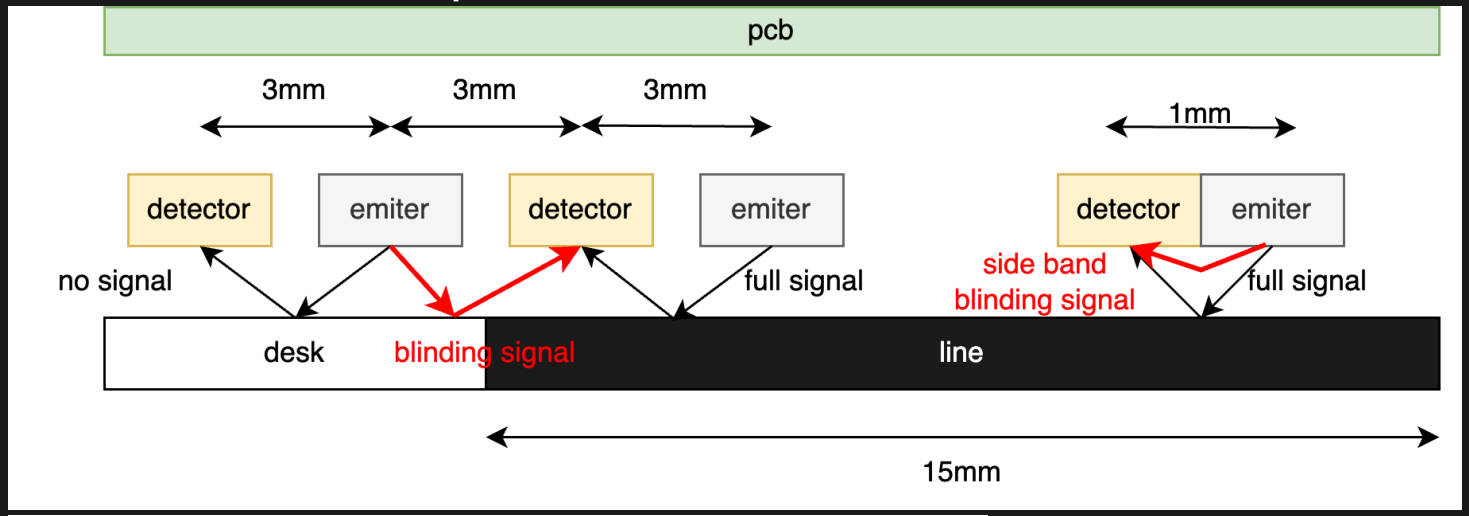


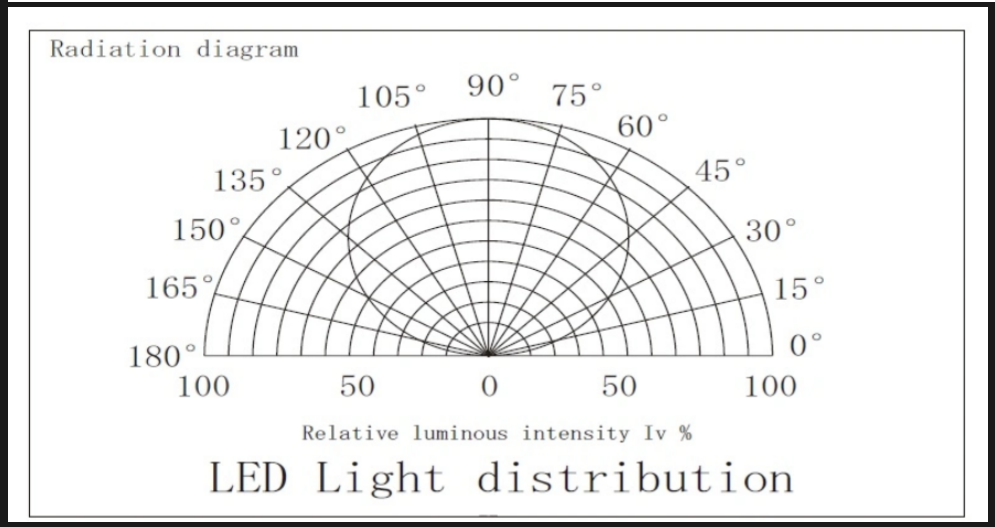
correct center sensors placement :



line sensors - geometry

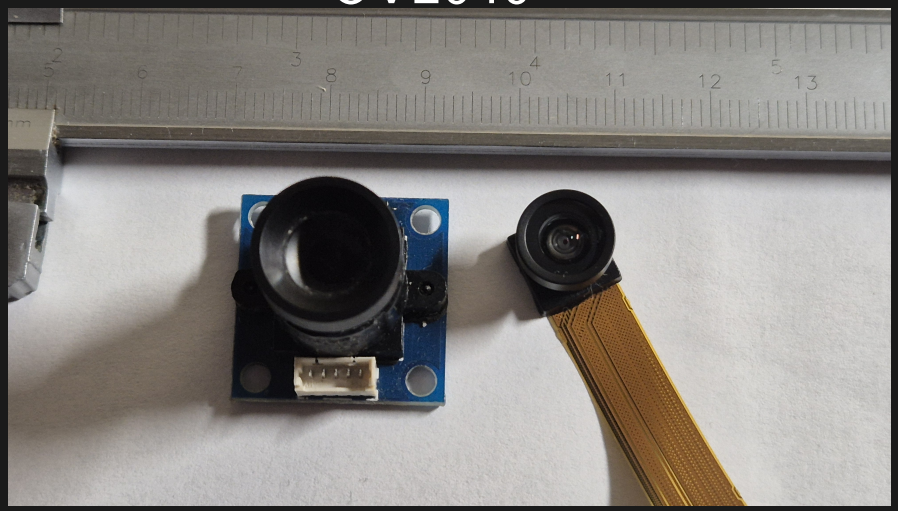
too close sensors placement :



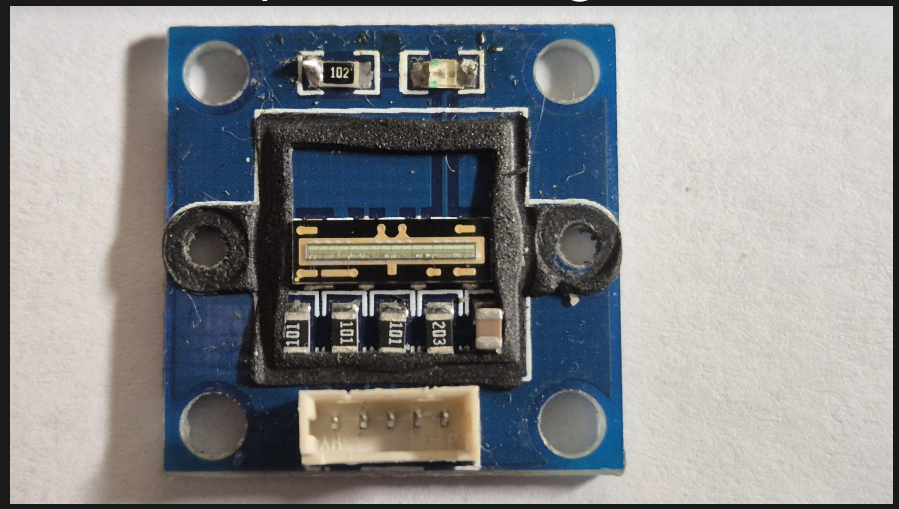


camera

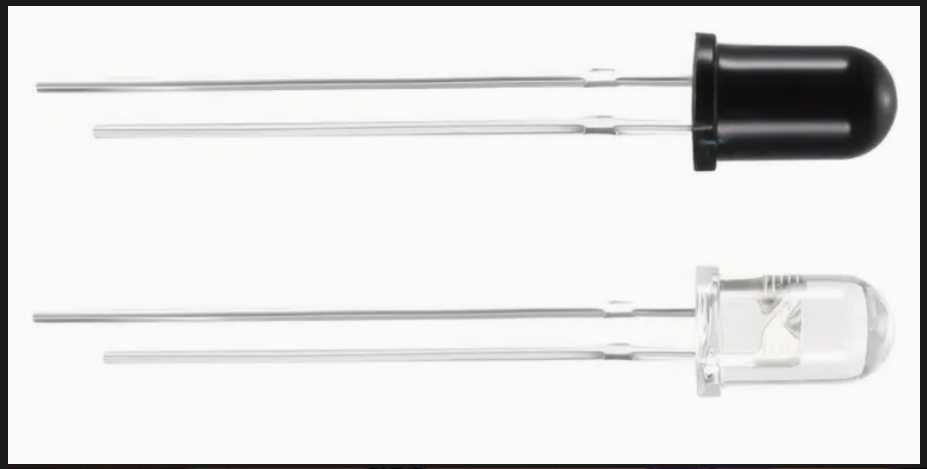
any DCMI smd camera, e.g. OV2640

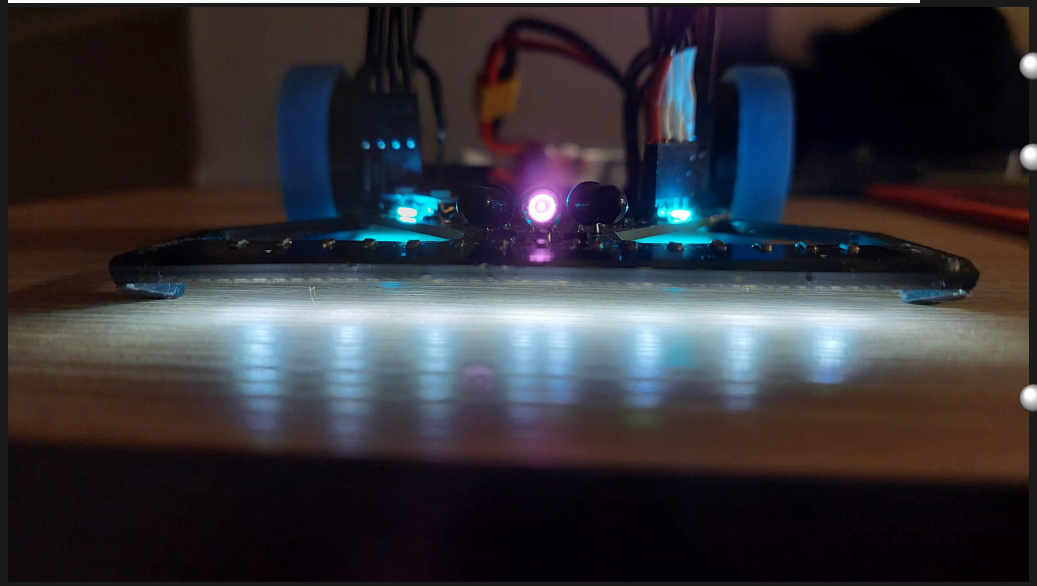


line scan camera, TSL1401CL, 128pixels, analog out



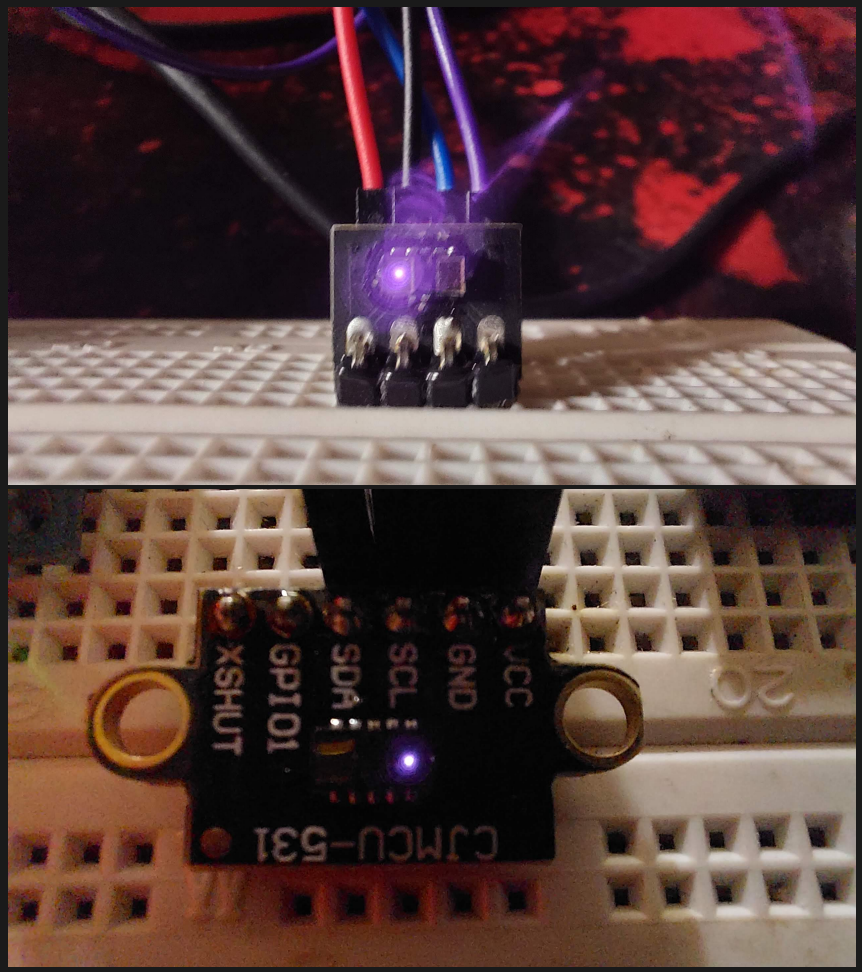
IR leds sensors





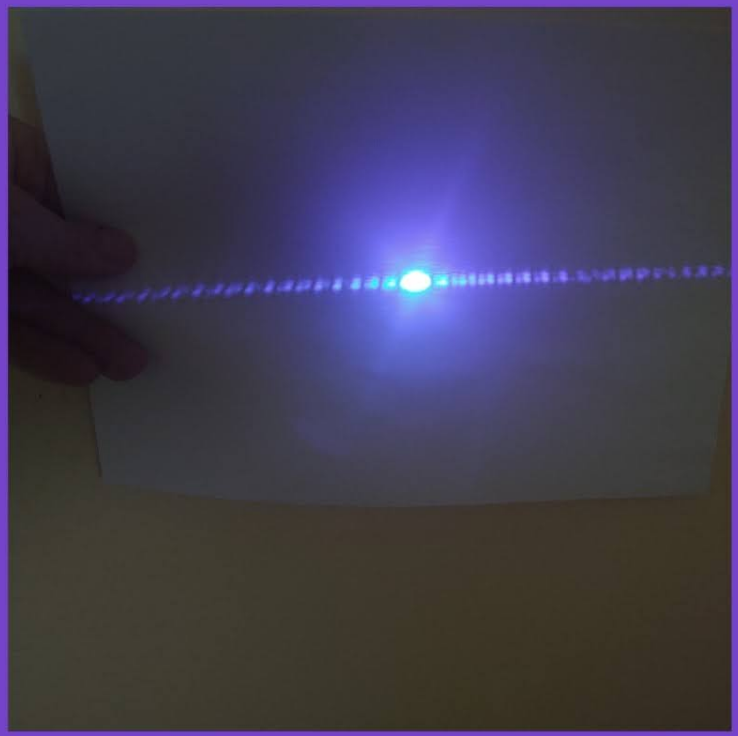
- turn off led
- measure voltage
- turn on led
- measure voltage
- substract
- put result into polynom (2nd .. 3rd order) to convert into distance
- put result into filter (low pass or Kalman)

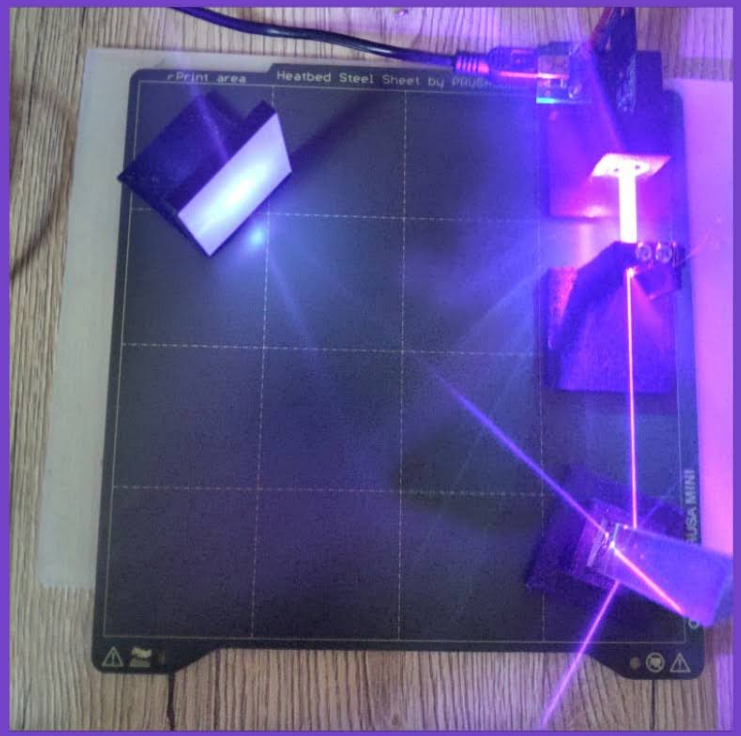
laser TOF

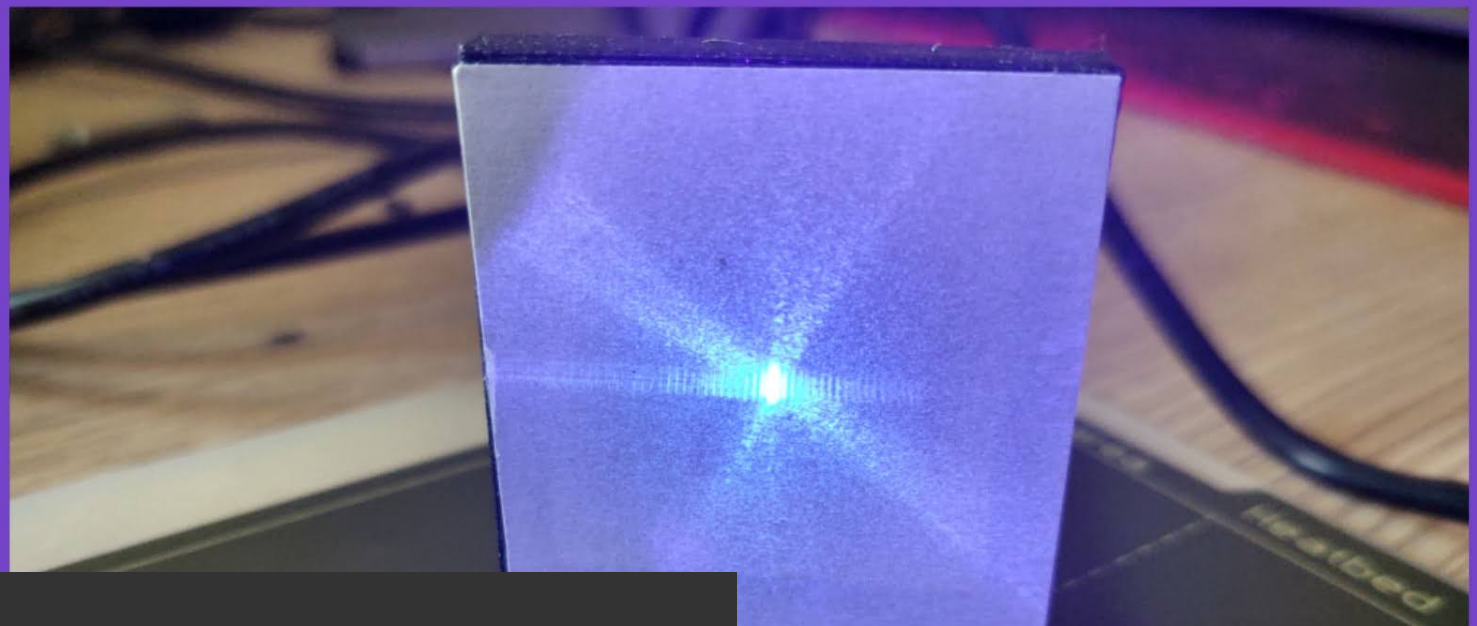


- **NEVER buy cheep alternative**
- TOF : VL53L0X, VL53L1CX
- TOF 8x8 : VL53L8CX
- LIDAR on a chip, 54 x 42 : VL53L9CA

laser

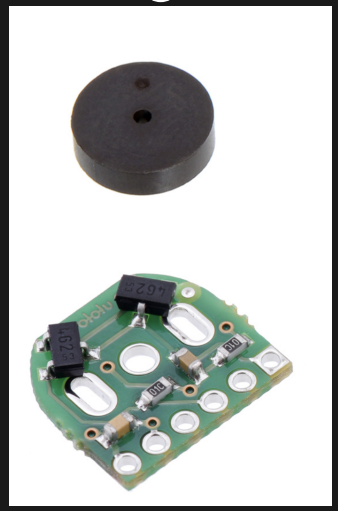




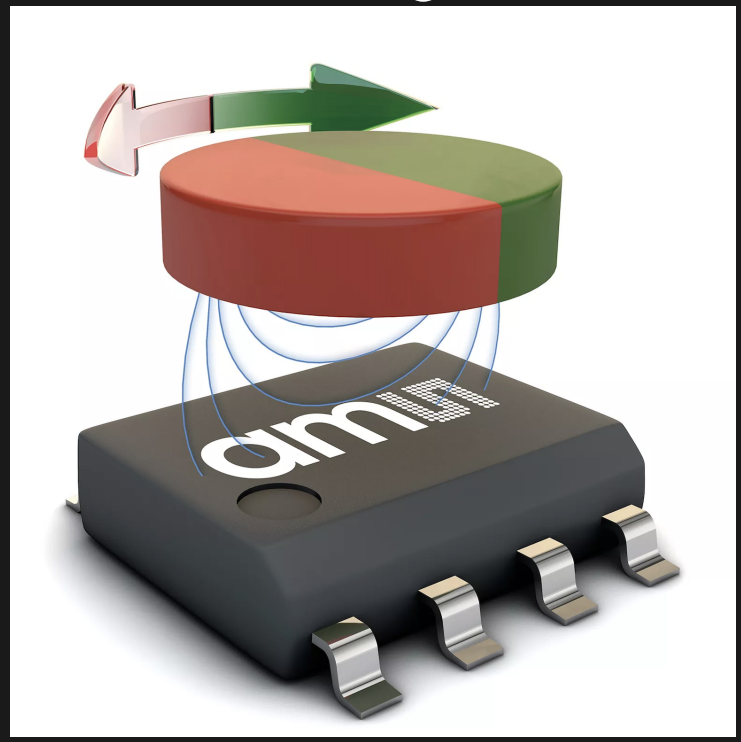


encoders

relative quadrature encoder,
resolution 3.6deg



absolute encoder,
resolution 0.087deg



robot controll

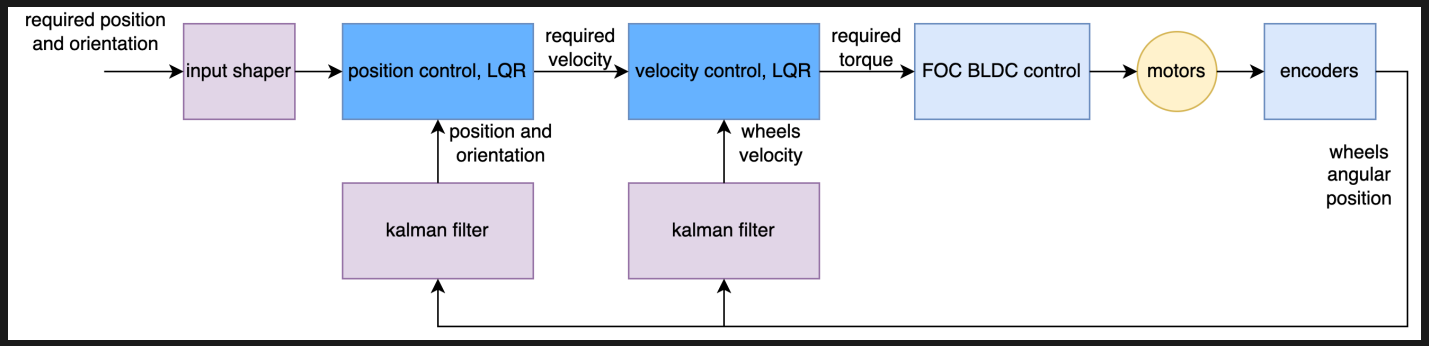
control levels :

- 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 :

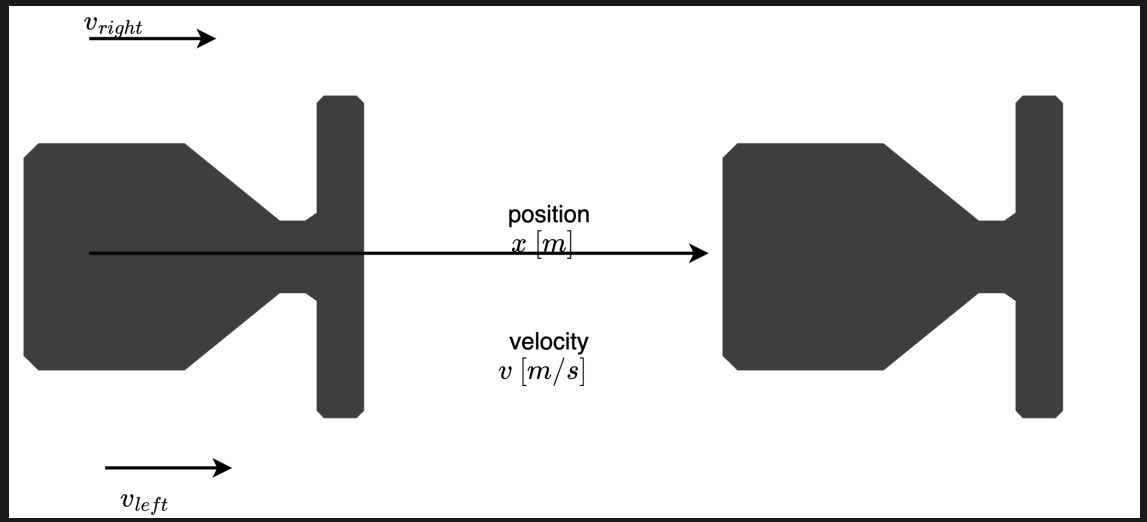
- if else : never captures dynamics
- PID : only SISO systems, only hand tunning
- LQR+Kalman filter : MIMO systems, exact solution
- MPC : MIMO systems, tracking (planning trajectory)
- neural network :RNN (LSTM or GRU), reinforcement learning

controller loops

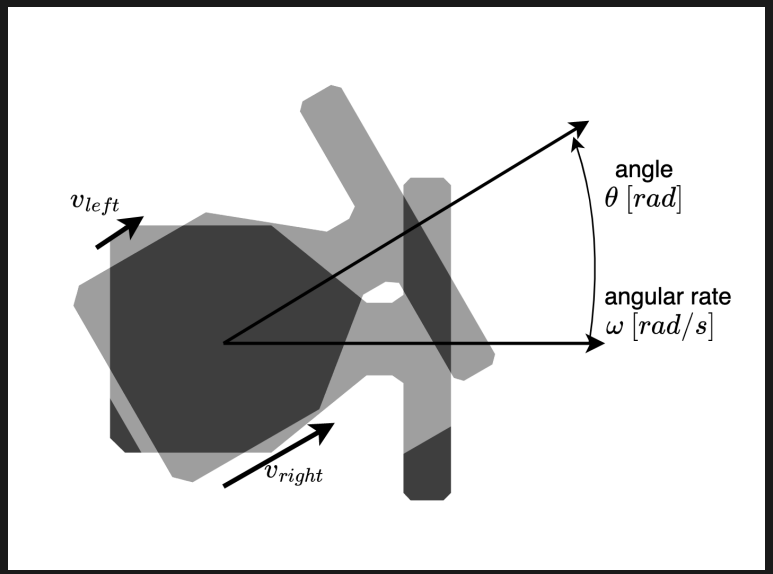


- if we dont know system dynamics (only guessing) :
  - **if else** controller
  - hand tunned **PID** controller
- from known dynamics :
  - Kalman filter
  - LQR, LQG, H∞
  - **MPC**, neural network - planning

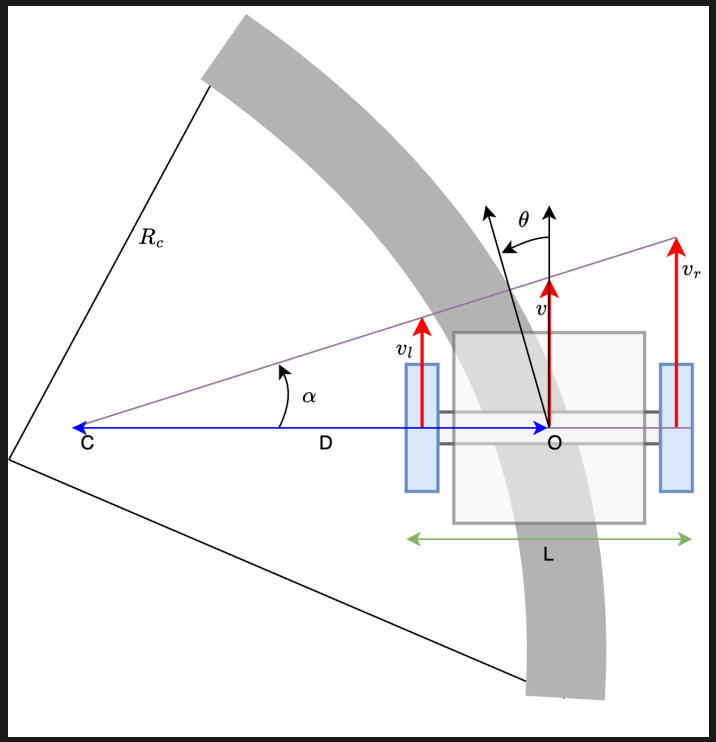
robot controll - translation



robot controll - rotation



robot controll - overview

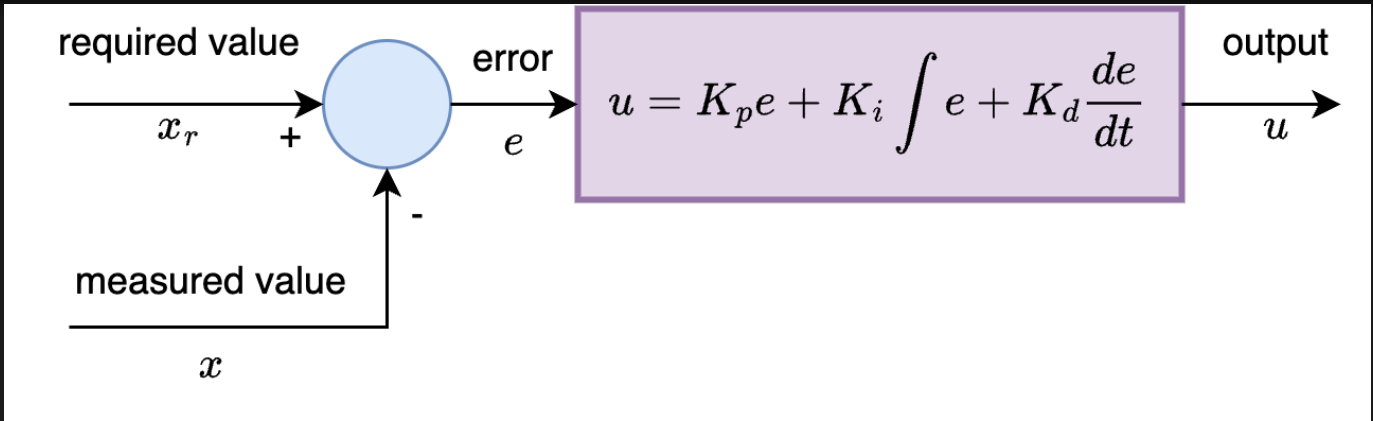


- line following is motion on circle
- estimate line curvature radius Rc
- estimate maximum velocity v
- control movement on circle with radius Rc and velocity v
- velocity is limited by motors saturation and centrifugal force

robot controll - overview

- position control : distance x and angle θ
- required values : x_r, θ_r
- observed values : x, θ
- forward : $x_r(n) = x(n) + speed_r$
- turning : $\theta_r(n) = \theta(n) + angular\_vel_r$
- full state vector : (x, θ, v, ω)
- note 1, : vectors are matrices, with shape Nx1
- note 2, : nice to have model of dynamics, **planning**

PID



discrete form:

$$u(n) = u(n-1) + k_0 e(n) + k_1 e(n-1) + k_2 e(n-2)$$

$$k_0 = k_p + k_i + k_d$$

$$k_1 = -k_p - 2k_d$$

$$k_2 = k_d$$

- antiwindup : clipping output
- error filter :

$$e_f(n) = \alpha e_f(n-1) + (1-\alpha)e(n)$$

PID - common mistakes

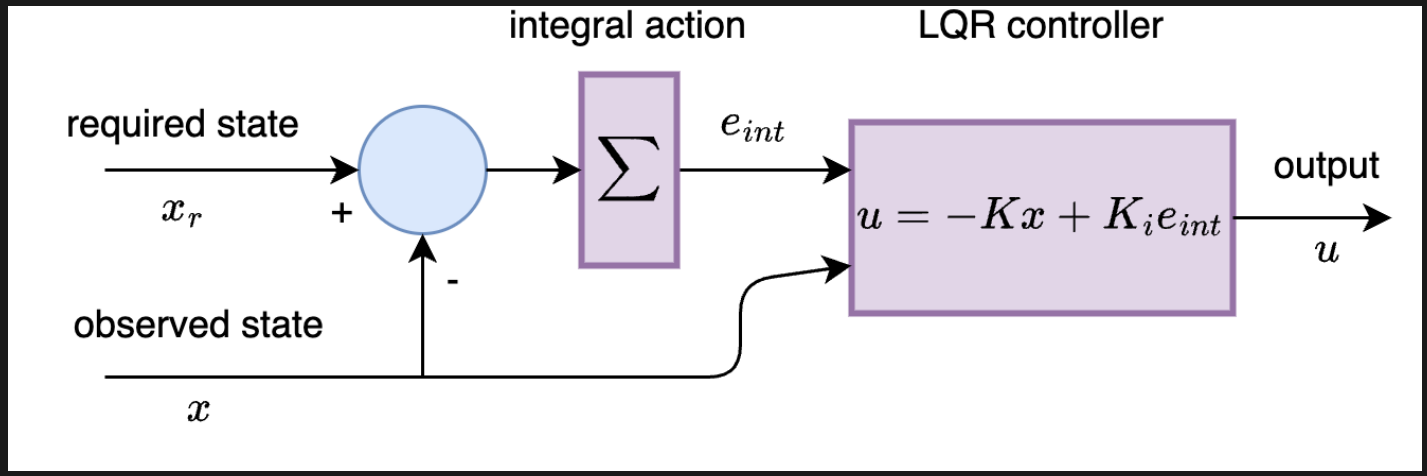
- no antiwindup, **real u is limited !!!**

$$e_{sum}(n) = e_{sum}(n-1) + e(n)$$
$$u_{int}(n) = K_i e_{sum}(n)$$

- no derivative filter, **real e is noised !!!**

$$d(n) = e(n) - e(n-1) + \epsilon(n) - \epsilon(n-1)$$
$$u_{der}(n) = K_d d(n)$$

LQR - linear quadratic regulator (optimal control)



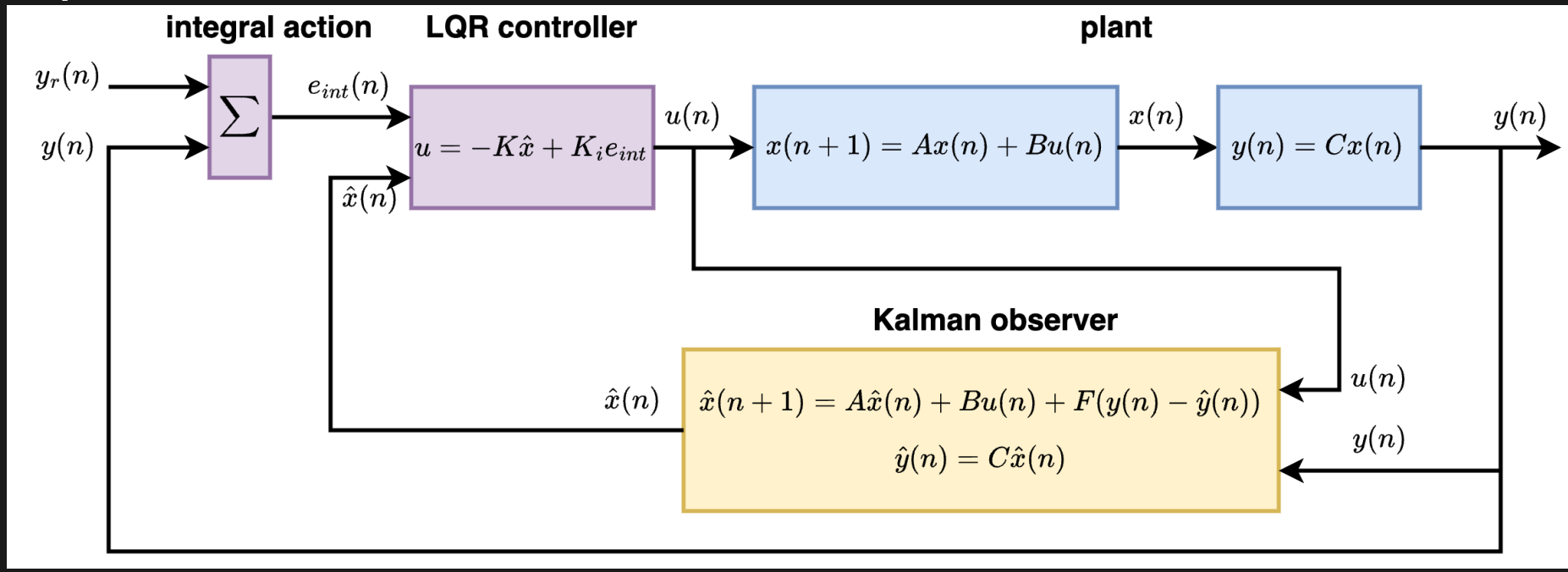
discrete form:

$$u(n) = -Kx(n) + K_i e_{int}(n)$$
$$e_{int}(n) = e_{int}(n-1) + x_r(n) - x(n)$$

- antiwindup : integral back-calculation
- error filter : use Kalman observer

LQG - linear quadratic gaussian control

LQR + Kalman observer

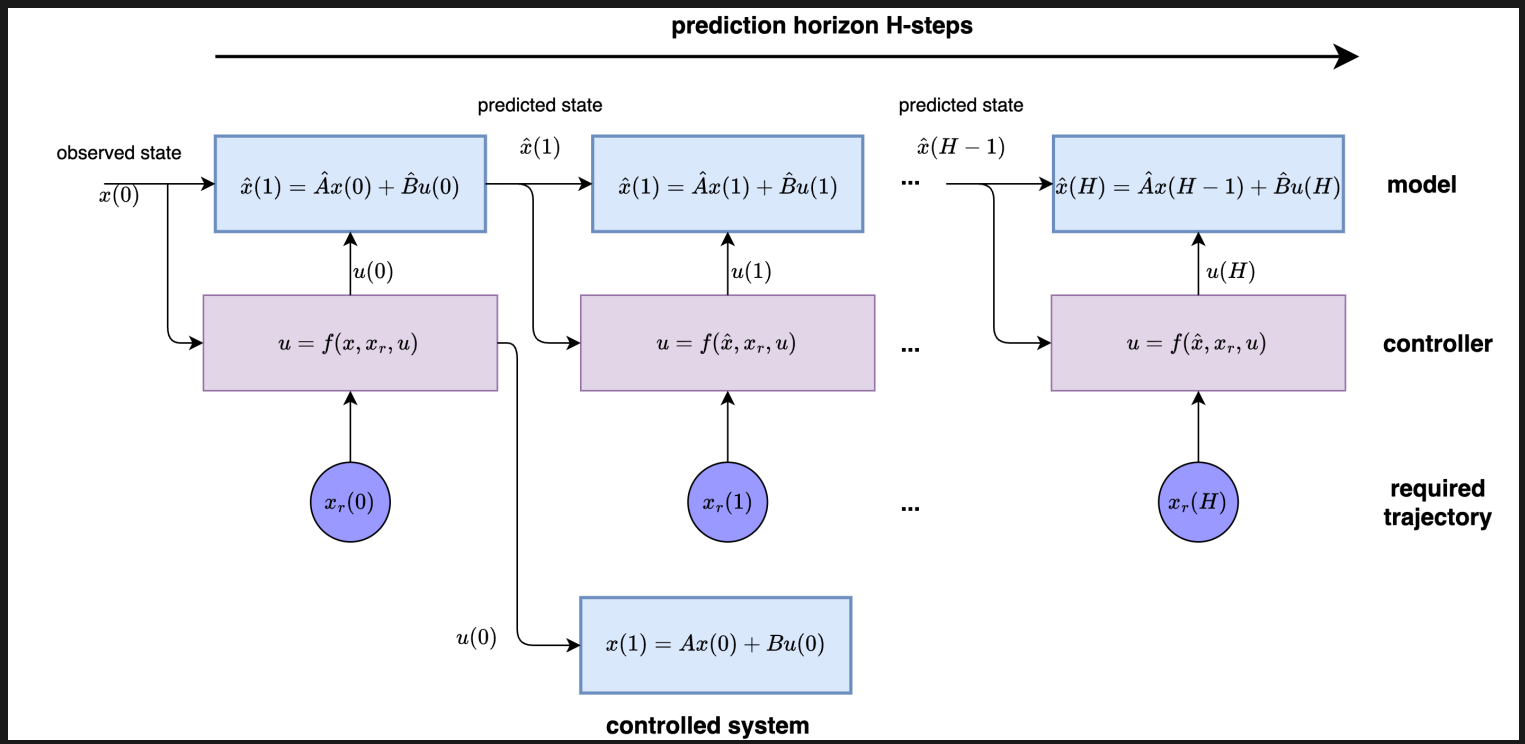


LQG - linear quadratic gaussian control

```
void step()
{
    // integral action
    auto error = this->yr - this->y;
    auto integral_action_new = this->integral_action + this->ki*error;

    //LQR controll law
    auto u_new = this->k*this->x_hat*(-1.0) + integral_action_new;

    //antiwindup with conditional integration
    this->u = u_new.clip(-antiwindup, antiwindup);
    this->integral_action = integral_action_new - (u_new - this->u);

    // kalman observer
    // only y is known, and using knowledge of dynamics,
    // the full state x_hat can be reconstructed
    auto prediction_error = this->y - this->c*this->x_hat;
    this->x_hat = this->a*this->x_hat + this->b*this->u + this->f*prediction_error;
}
```

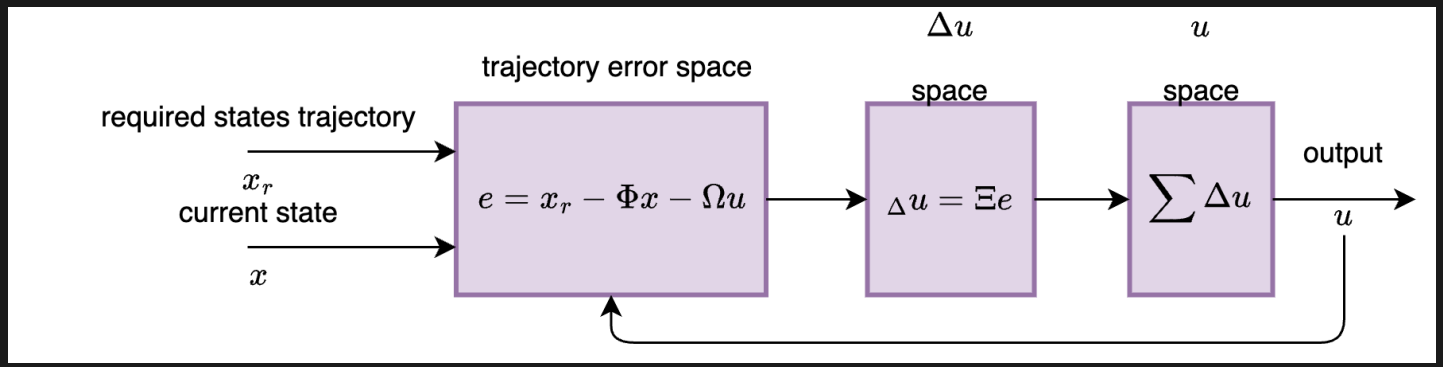
MPC - model predictive control

- not only regulating but full trajectory tracking
- planning multiple steps into futur
- x_r is now full trajectory (e.g. 50 steps) of states



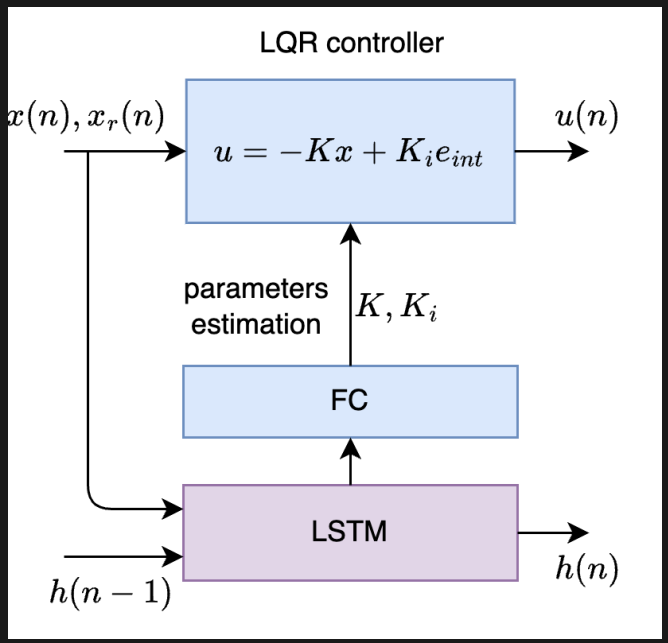
MPC - model predictive control

- not only regulating but full trajectory tracking
- planning multiple steps into futur
- x_r is now full trajectory (e.g. 50 steps) of states



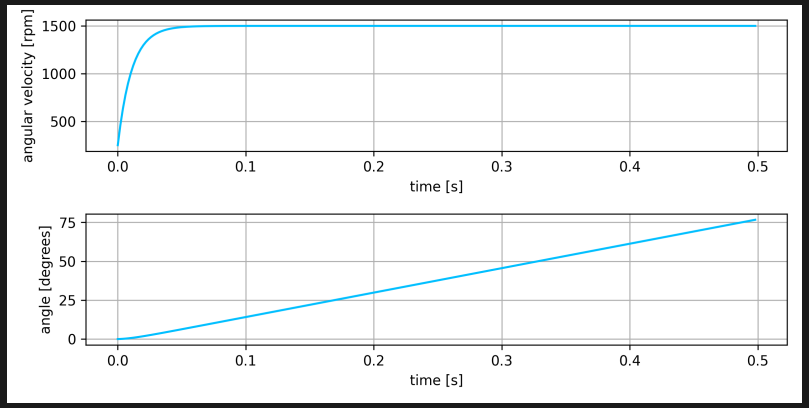
neural network high level control

- recurrent neural network (GRU, LSTM)
- RNN estimates controller matrices
- direct controll makes no sense

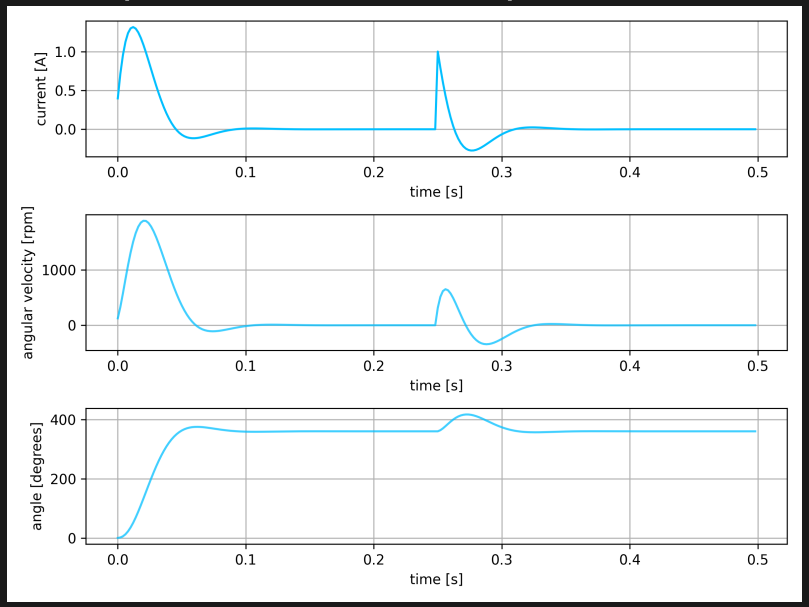


toy example - servo controll

open loop response :



closed loop response (LQR controller) :



servo state space model

1st order motor velocity model

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

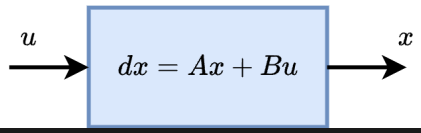
wheel angle

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

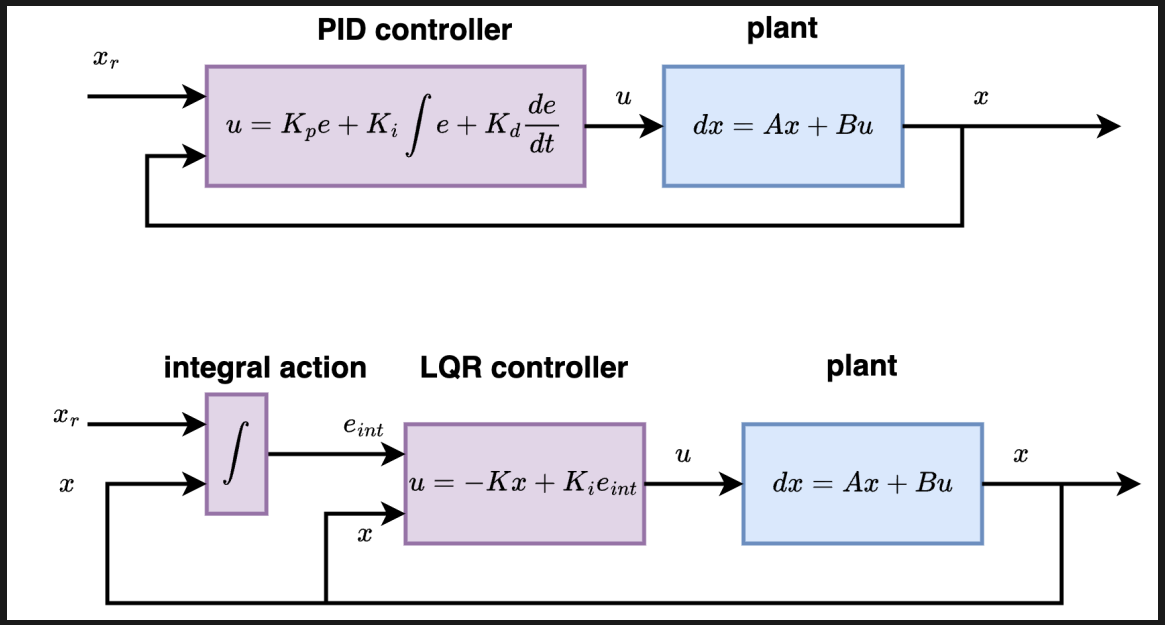
state space model

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

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



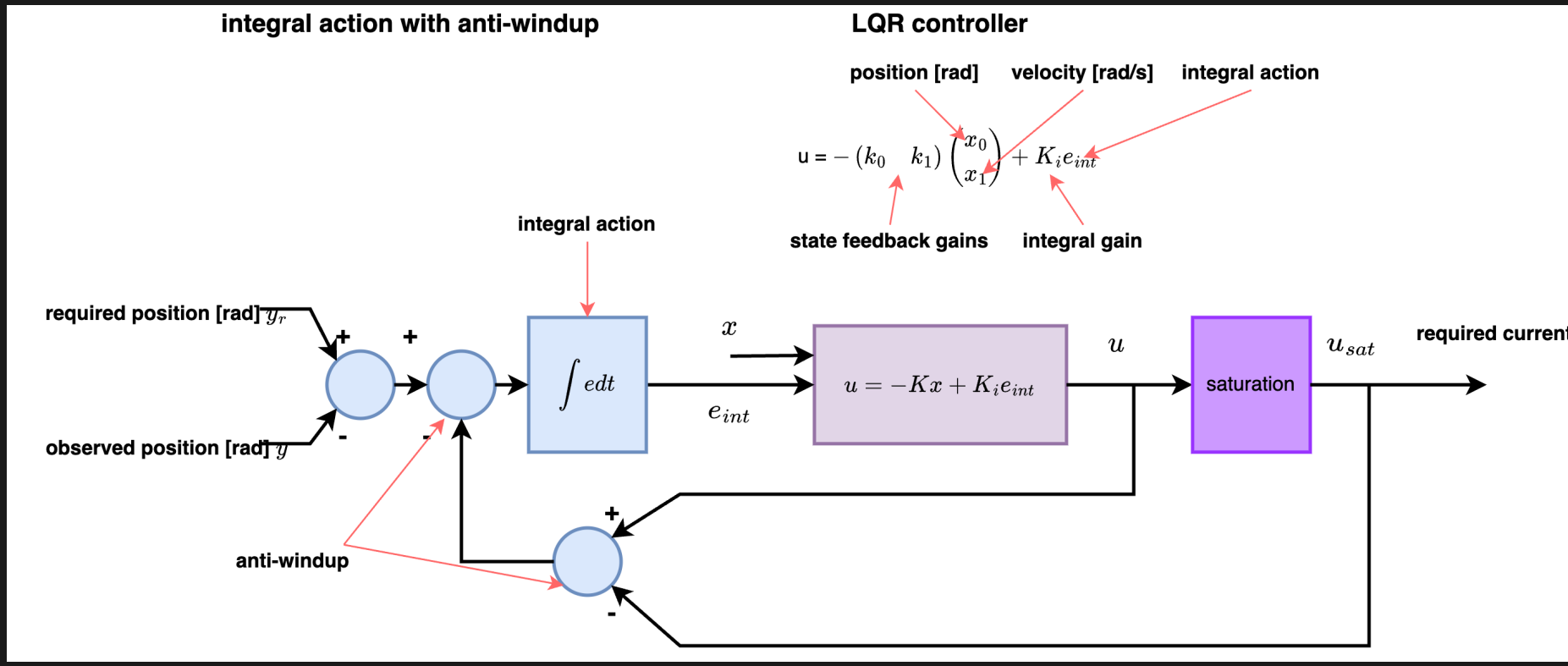
feedback loop



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

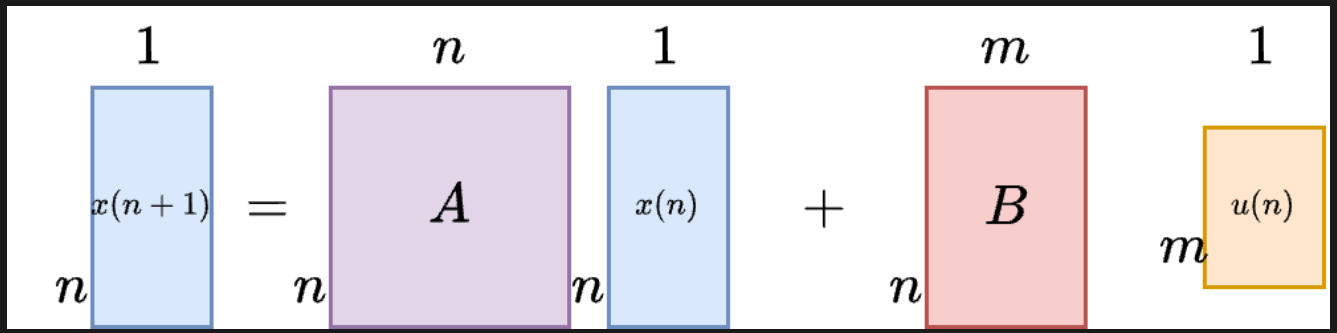
note : continuos modeling (and controllers) have slow inference time - requires Runge-Kutta solvers, always use discrete models and controllers

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 : $u = -Kx + K_i e_{int}$

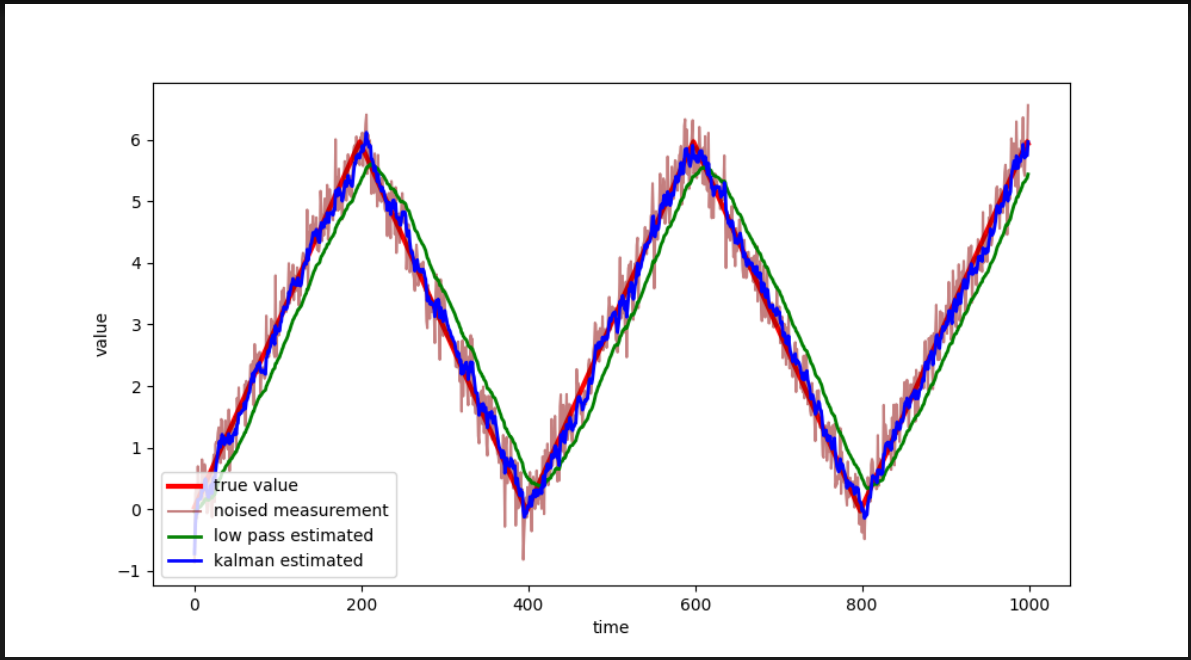
discrete linear state space model



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

- 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

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

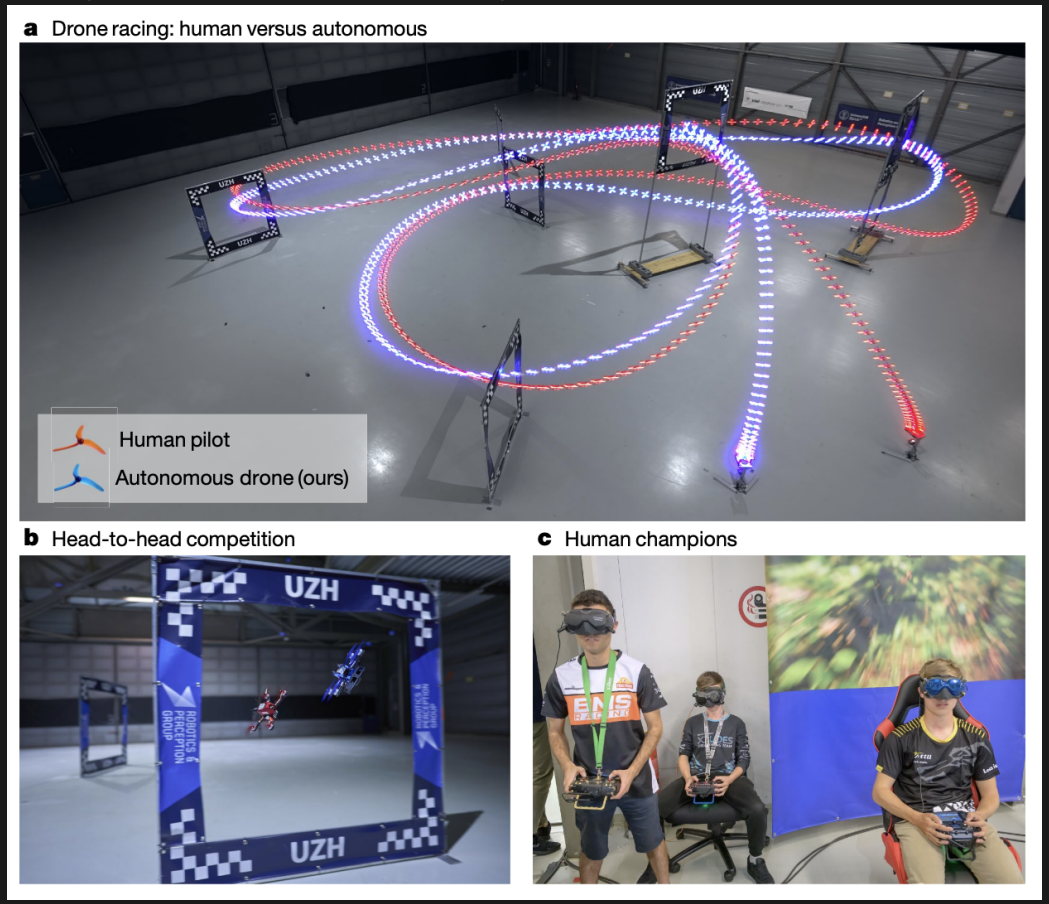
recommended sources

shops

- motors, wheels : `www.pololu.com`, `www.jsumo.com`
- PCB : `jlcpcb.com`
- electronics : `www.mouser.com`

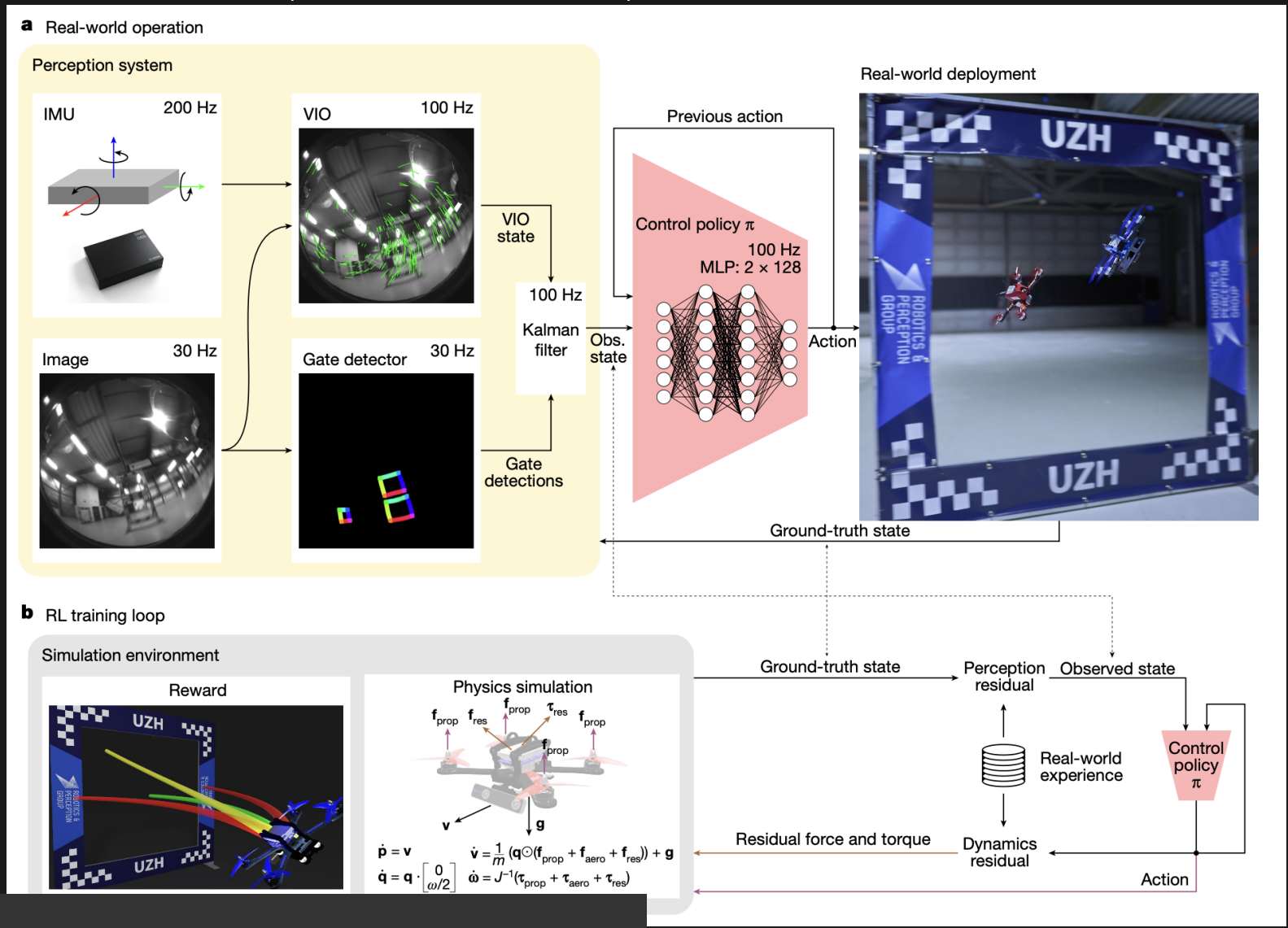
recommended sources

Champion-level drone racing using deep reinforcement learning, Elia Kaufmann1, Leonard Bauersfeld1, Antonio Loquercio1, Matthias Müller, Vladlen Koltun, Davide Scaramuzza



recommended sources

Champion-level drone racing using deep reinforcement learning, Elia Kaufmann1, Leonard Bauersfeld1, Antonio Loquercio1, Matthias Müller, Vladlen Koltun, Davide Scaramuzza



recommended sources

books

- The Matrix Cookbook, Kaare Brandt Petersen Michael Syskind Pedersen, https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf
- Kalman Filter from the Ground up, Alex Becker, www.kalmanfilter.net

groups

- Model Predictive Control (MPC) Tutorial, https://aleksandarhaber.com/model-predictive-control-mpc-tutorial-1-unconstrained-
- ETH Model predictive control group, ORCA project https://control.ee.ethz.ch/research/team-projects/autonomous-rc-car-racing/videos-related-to-the-orca-project.html https://www.youtube.com/watch?v=mXaElWYQKC4

Q&A

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

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

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

$$where \quad P(\infty) = Q$$

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

system identification

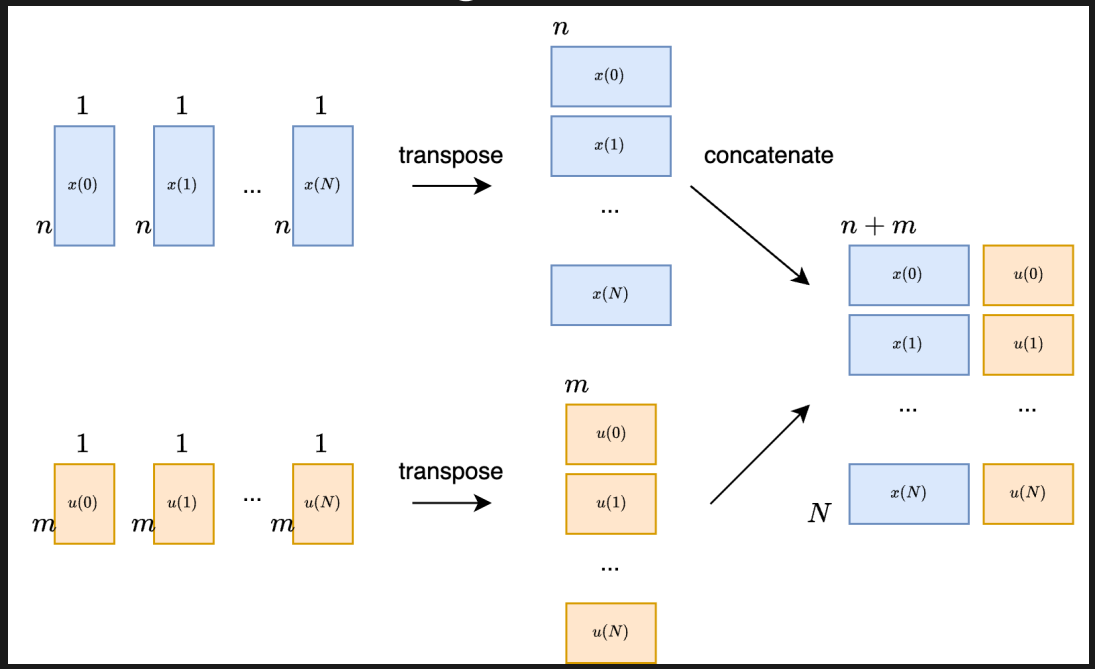
- generate input sequence : u
- measure data : x
- fitting model $\hat{A}$, $\hat{B}$ into data
- predicted data : $\hat{x}$

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

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

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

model predictive control

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

$$\mathcal{L} = \sum_{h=0}^{H-1} (x_r^T(h) - x^T(h)) Q (x_r(h) - x(h)) +_\Delta u^T(h) R_\Delta u(h)$$

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

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

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 $x(n)$ and $u(n-1)$ are used to predict all other x

$$
\begin{aligned}
x(n+1) &= Ax(n) + Bu(n) \\
&= Ax(n) + B(u(n-1) +_\Delta u(n)) \\
x(n+2) &= Ax(n+1) + B(u(n) +_\Delta u(n+1)) \\
&= A^2x(n) + (AB+B)u(n-1) + (AB+B)_\Delta u(n) + B_\Delta u(u) \\
x(n+3) &= Ax(n+2) + B(u(n+1) +_\Delta u(n+2)) \\
&= A^3x(n) + (A^2+AB+B)u(n-1) \\
&\quad + (A^2B+AB+B)_\Delta u(n) \\
&\quad + (AB+B)_\Delta u(n+1) + B_\Delta u(n+2) \\
&\quad \ldots
\end{aligned}
$$

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

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

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

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

we get

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

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

$$\frac{\partial \mathcal{L}}{\partial {}_{\Delta}\tilde{U}} = 2\tilde{R}\,{}_{\Delta}\tilde{U} - 2\Theta^T\tilde{Q}S + 2\Theta^T\tilde{Q}\Theta\,{}_{\Delta}\tilde{U}$$

$$0 = 2\tilde{R}\,{}_{\Delta}\tilde{U} - 2\Theta^T\tilde{Q}S + 2\Theta^T\tilde{Q}\Theta\,{}_{\Delta}\tilde{U}$$

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

and obtain solution for model predictive controll

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

model predictive control - full algorithm

given matrices :

$$\tilde{Q}, \tilde{R}, \Theta, \Phi, \Omega$$

initialization (precompute) :

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

in loop :

$$\begin{aligned} E(n) &= X_r(n) - \Phi x(n) - \Omega u(n-1) \\ {}_{\Delta}u(n) &= \Xi E(n) \\ u(n) &= u(n-1) +_{\Delta} u(n) \end{aligned}$$