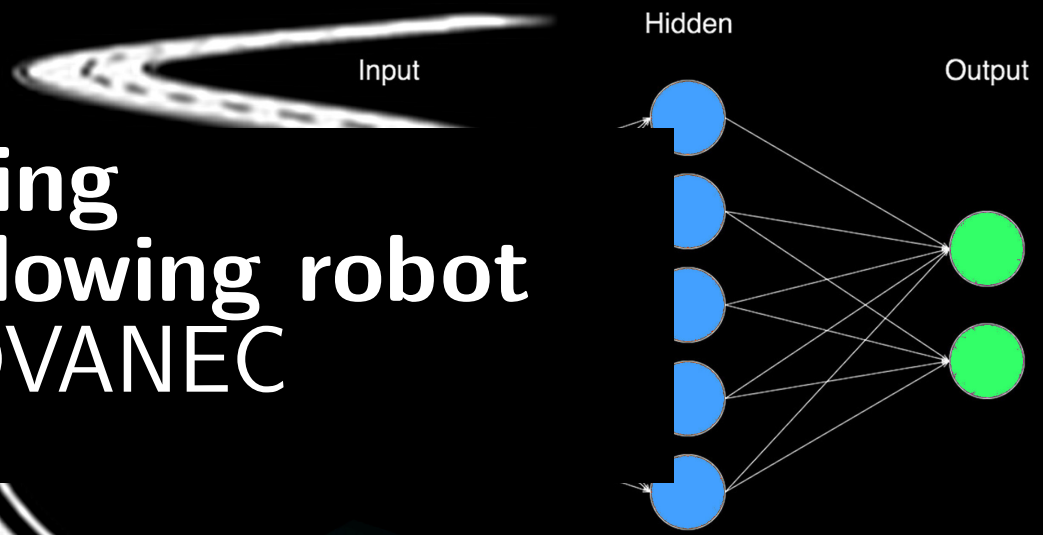
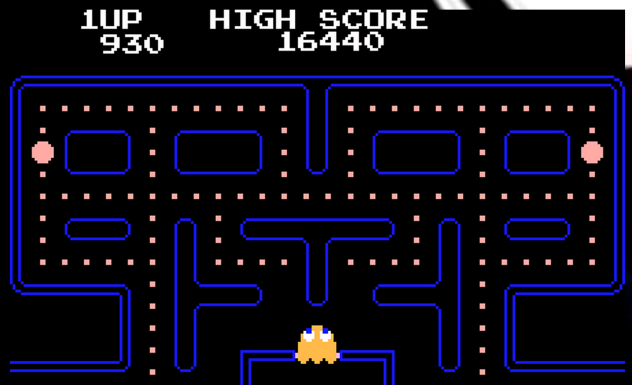
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \lambda \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

$$\left(\text{The New Action Value} = \text{The Old Value}\right) + \text{The Learning Rate} \times \left(\text{The New Information} - \text{the Old Information}\right)$$

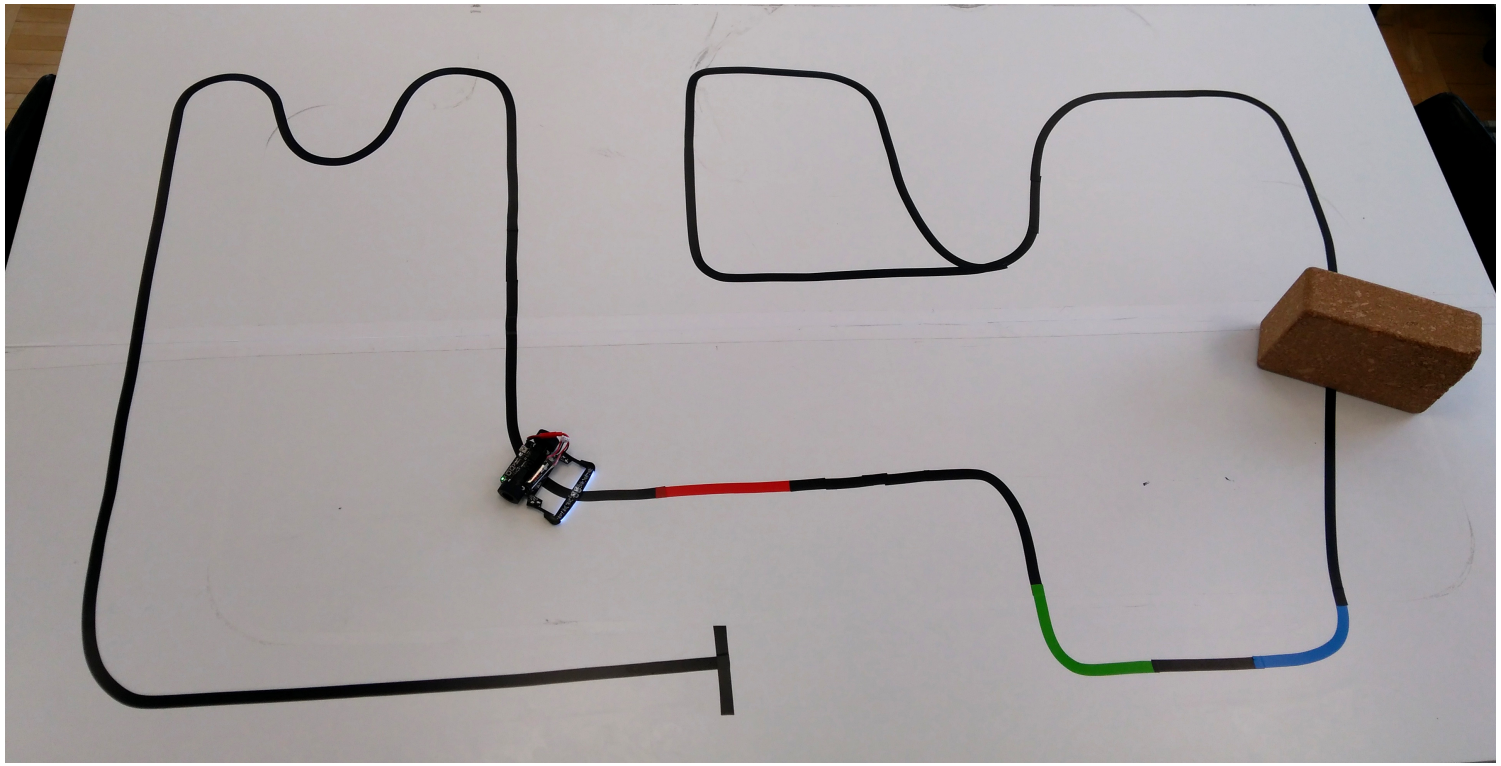
deep learning for line following robot

Michal CHOVANEC





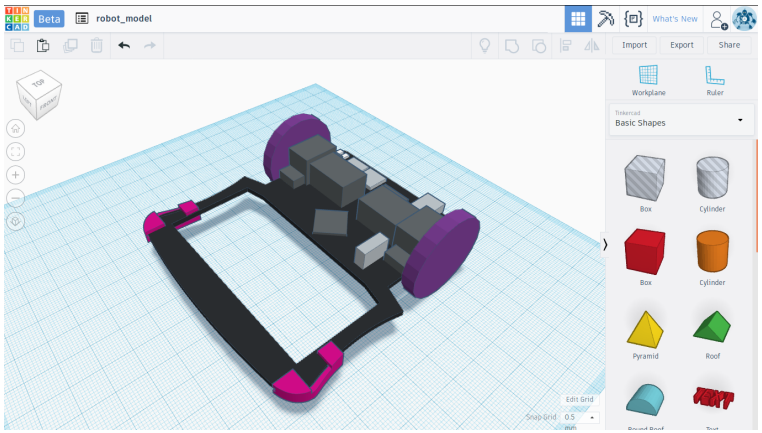
line follower competition

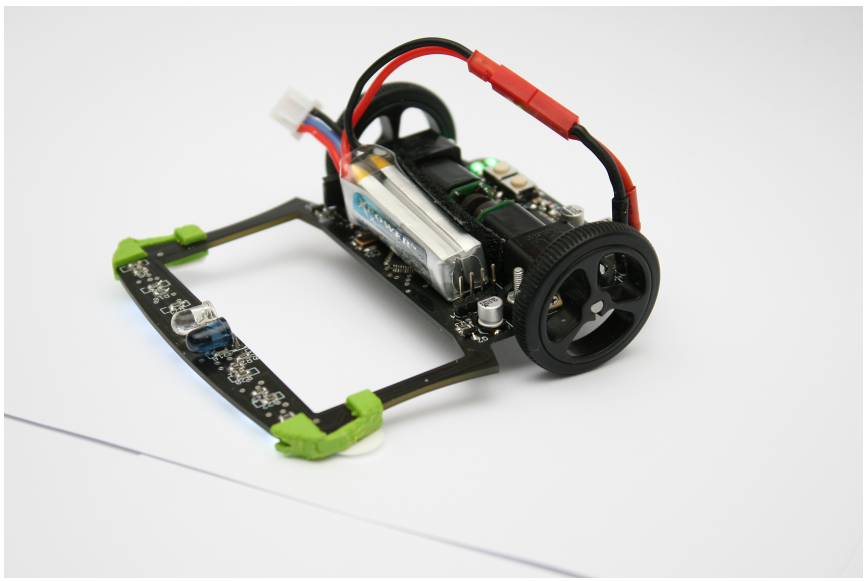


- SR - Istrobot
- CR - Roboticky den
- AU - Robot Challenge
- PL - Zawody robotow

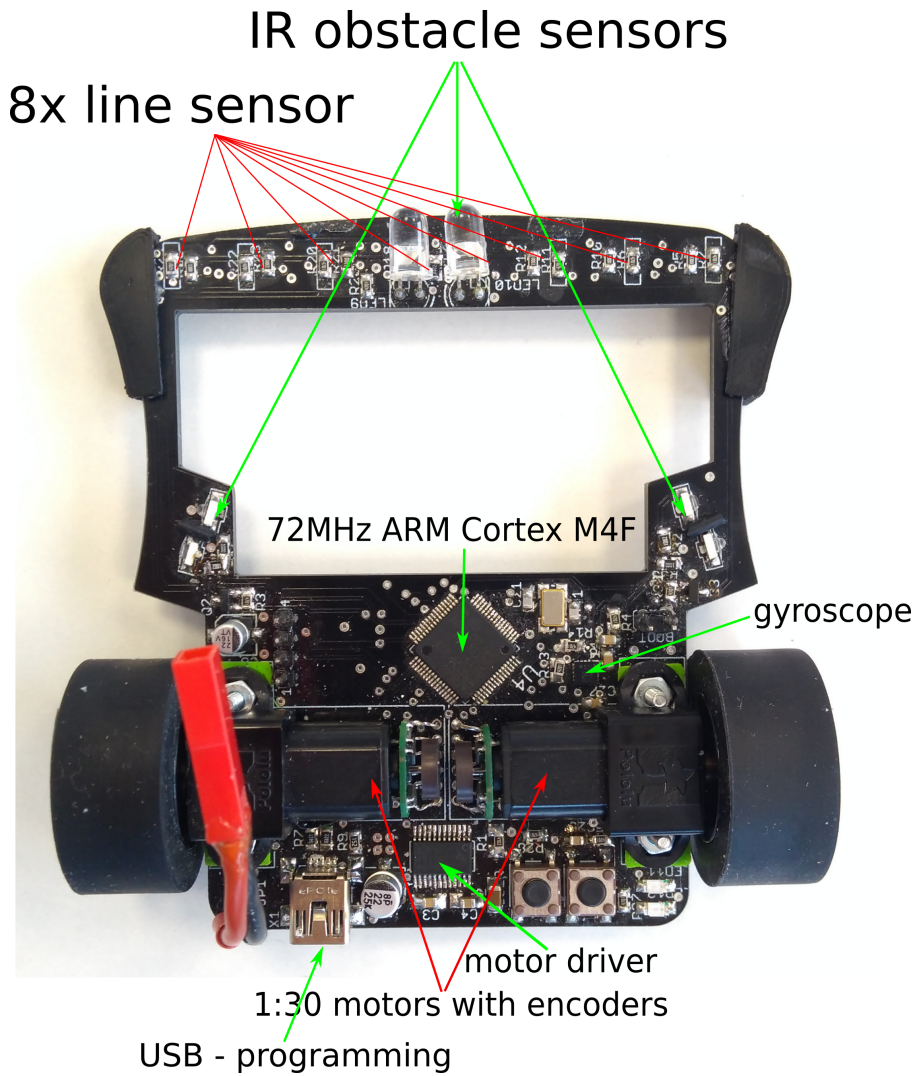
what does it take

- Hardware
  - strong light motors
  - high adhesion tyres
  - light accumulator
  - fast CPU
  - dozen of sensors
- Software
  - hard real time OS
  - tunned PIDs
  - predictive controll
  - mapping



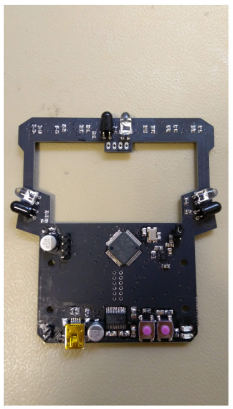


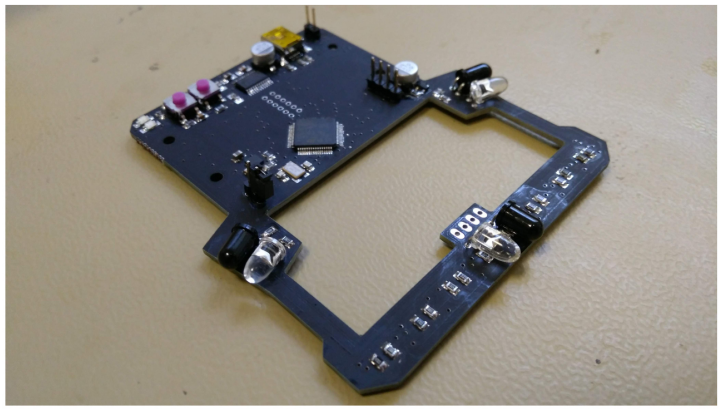
motoko uprising - hardware

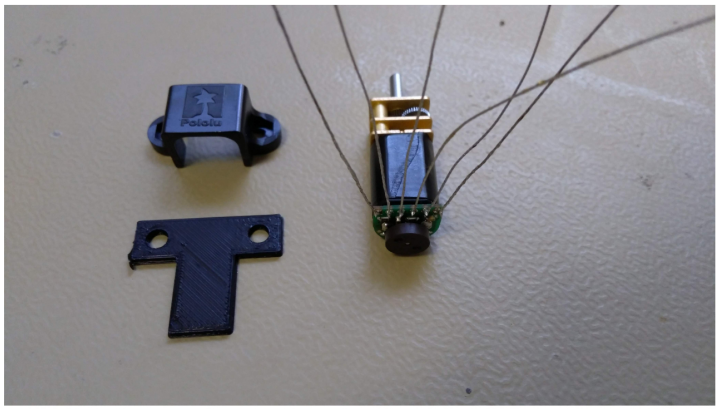


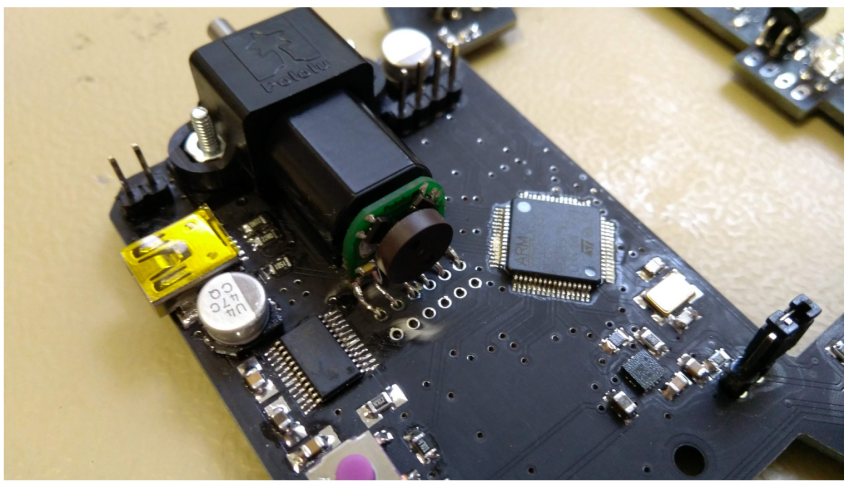
- **mcu** : STM32F303, 72MHz ARM Cortex M4F
- **motor driver** : TI DRV8834
- **motors** : 1:30 HP Pololu, with magnetic encoder
- **tyres** : Pololu 28mm diameter
- **line sensor** : 8x 540nm phototransistor + white LED
- **obstacle sensor** : SMD IR phototransistor + IR LED
- **imu** : LSM6DS0, gyroscope + accelerometer

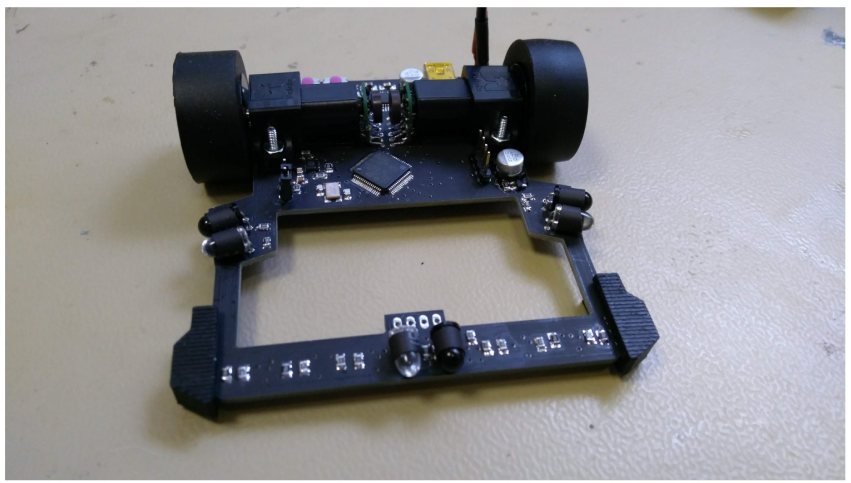
motoko uprising - hardware





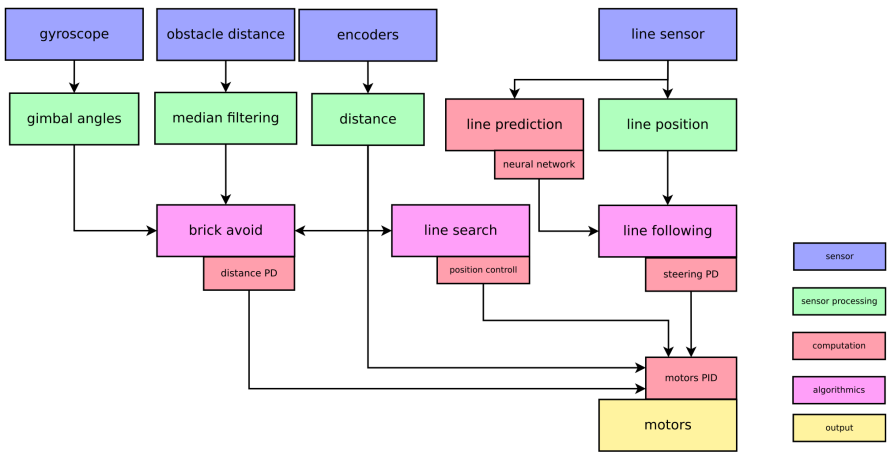




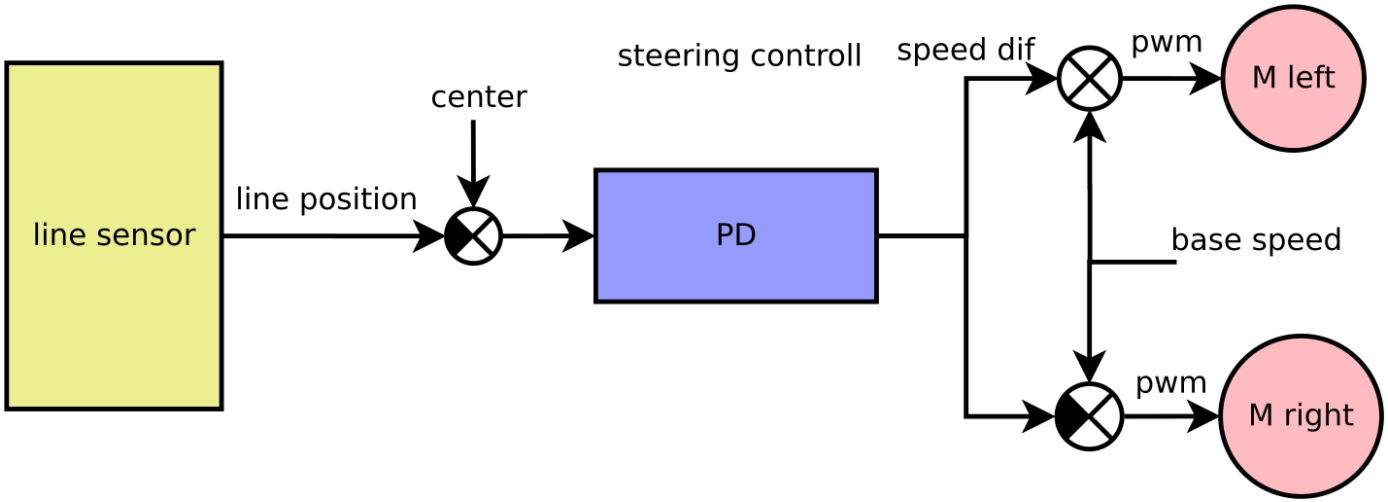


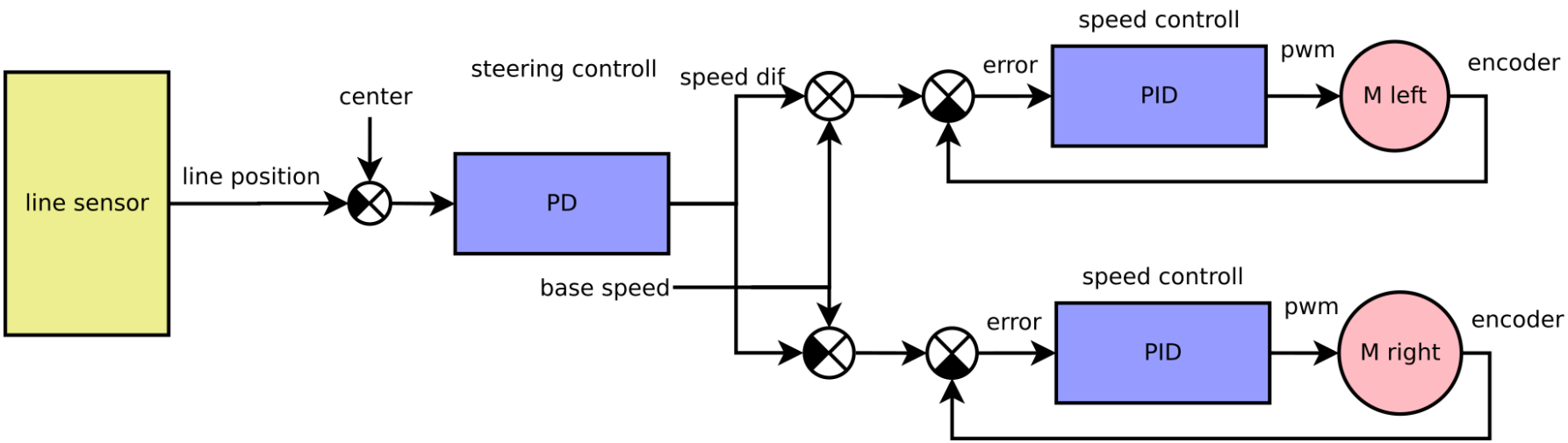
software

- **quadratic interpolation** : high precission line position computing
- **steering PID** : PD controller for steering
- **motor PID** : two PIDs for motors speed controll
- **curve shape prediction**
  - **fast** run on straight line, **brake** on curve
  - **deep neural network**
- written in **C++**
- **network training** on GPU - own framework for CNN

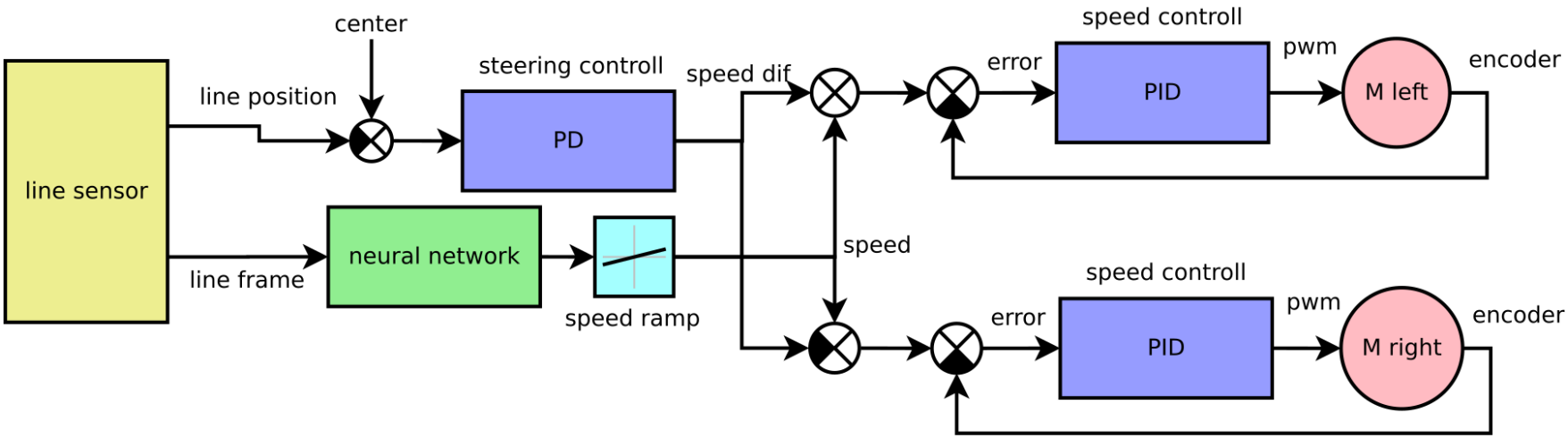


controll loop(s)



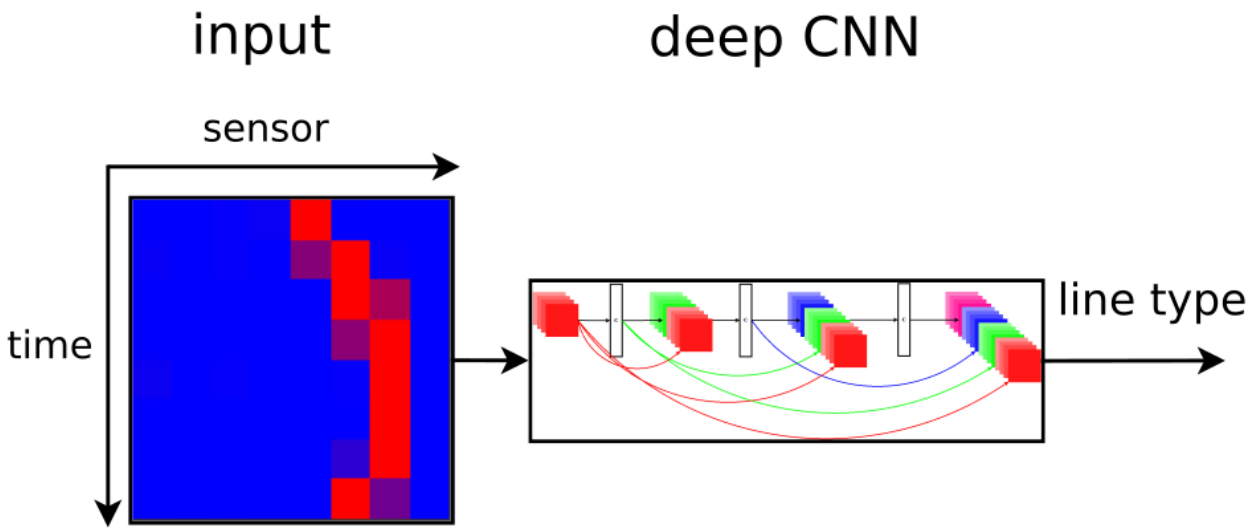


controll loop(s)

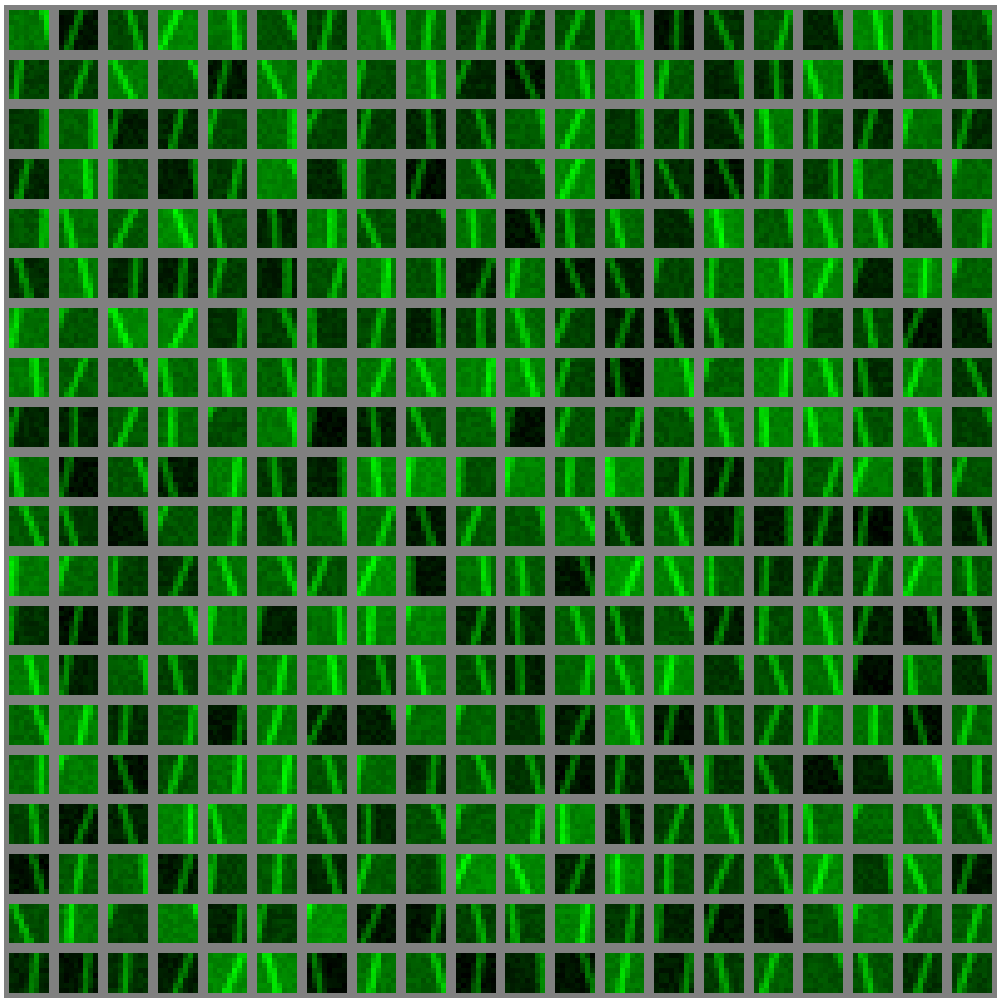


line shape prediction

- **fast** run on straight line, **brake** on curve
- **neural network** for line type classification
  - DenseNet - densely connected convolutional neural network
- **input** 8x8 matrix raw data from line sensors
  - 8 past line positions from 8 sensors
- **output** 5 curves types

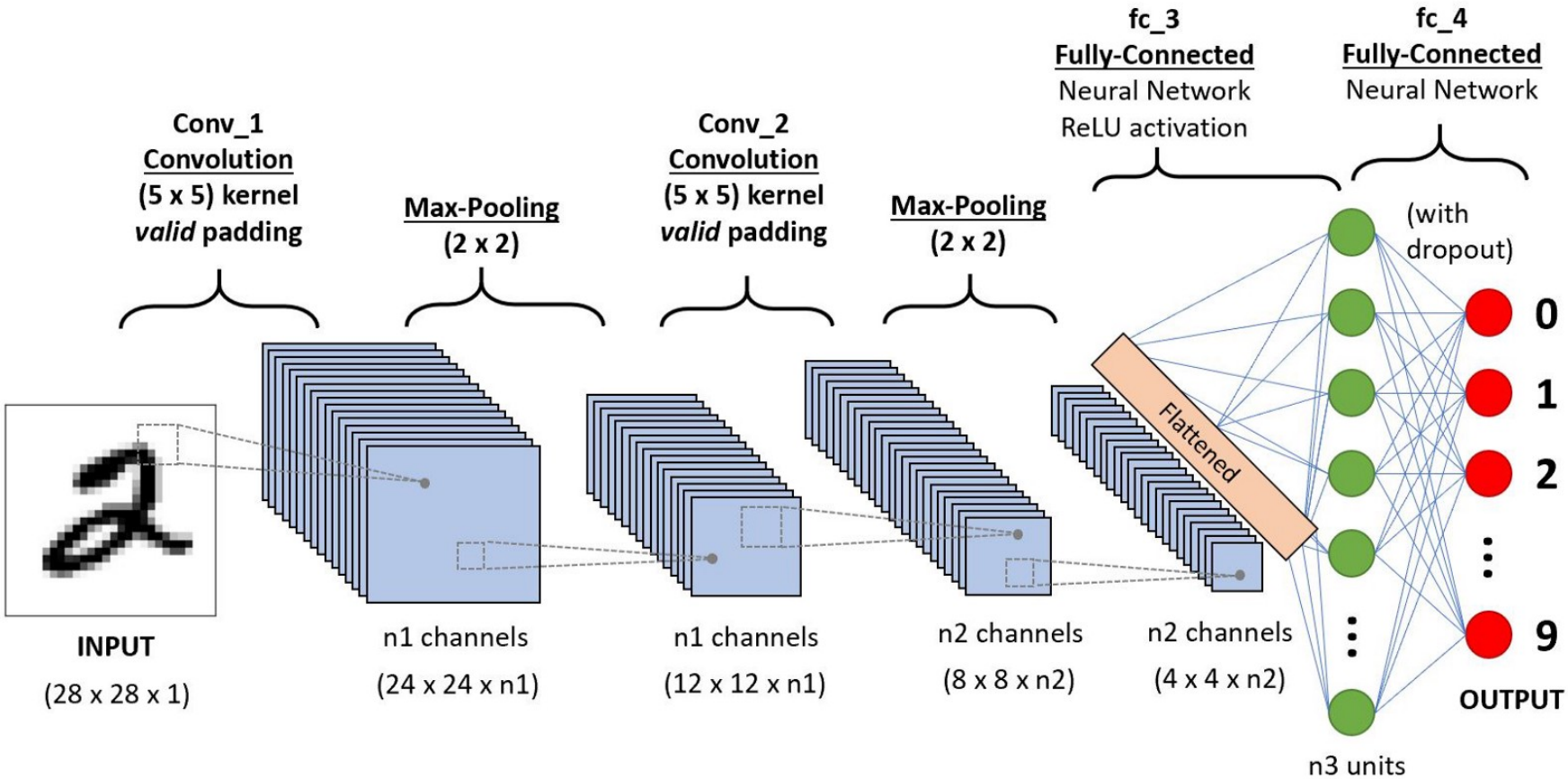


line dataset

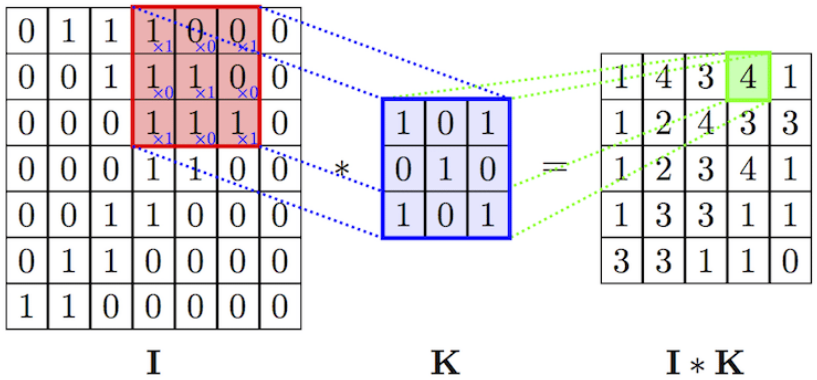


- 20000 for training
- 2500 for testing
- 8x8 inputs
- 5 outputs (5 curves types)
  - two left
  - one straight
  - two right
- augmentation - luma noise, white noise

convolutional neural network - CNN



discrete convolution



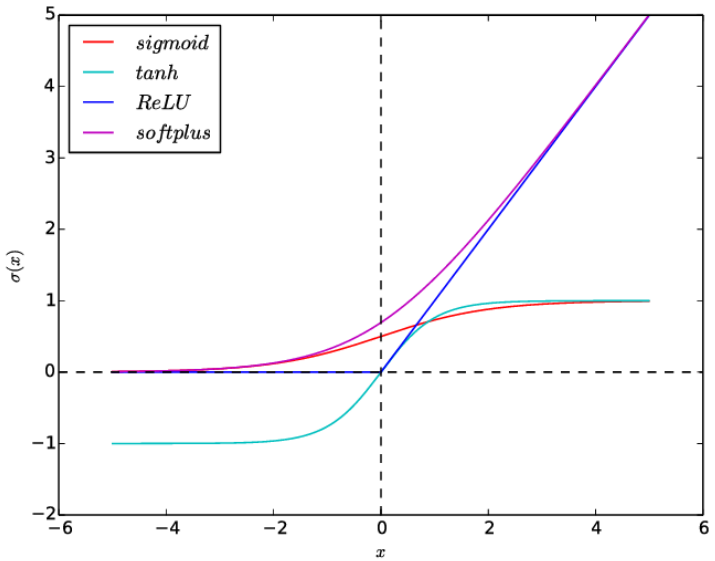

```
for (unsigned y = 0; y < input_height; y++)
for (unsigned x = 0; x < input_width; x++)
{
    float sum = 0.0;

    for (unsigned ky = 0; ky < kernel_height; ky++)
    for (unsigned kx = 0; kx < kernel_width; kx++)
    {
      sum+= kernel[ky][kx]*input[y + ky][x + kx];
    }

    output[y + kernel_height/2][x + kernel_width/2] = sum;
}
```

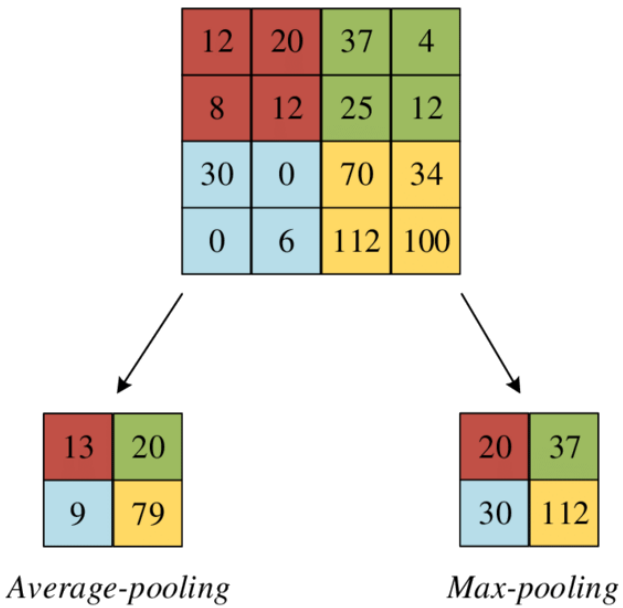
activation function

tanh, sigmoid, gaussian, **RELU**, leaky RELU ...



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \qquad \frac{df(x)}{dx} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

pooling


12 20 37 4
8 12 25 12
30 0 70 34
0 6 112 100
13 20
9 79
20 37
30 112
Average-pooling
Max-pooling

target hardware

target	bits	features	frequency
AVR Atmega 328	8	single cycle ADD, MUL	20MHz
ARM Cortex M0	32	single cycle ADD, MUL	48MHz
ARM Cortex M4, M7	32	SIMD DUAL 16bit MAC , FPU ...	72MHz 216MHz

embedded network implementation

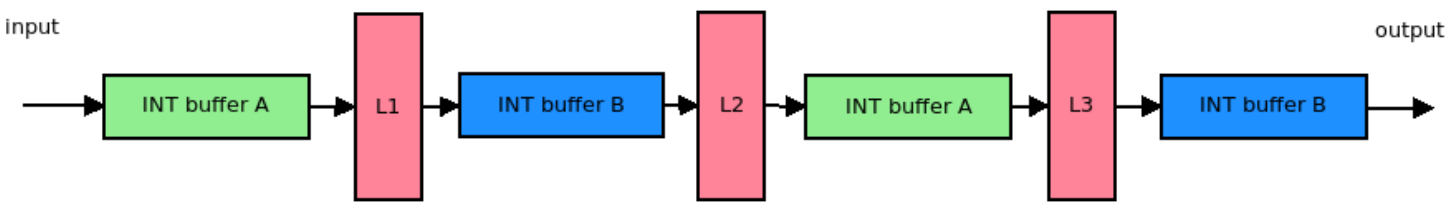
Quantization : convert float weights to int8_t

$$scale = max(|\vec{w}|_1)$$

$$\vec{w}' = \vec{w}\frac{127}{scale}$$

Memory saving : use double buffer memory trick

- unsigned buffer_size = $\max_i$(layers[i].input_size());
- buffer_a = new int8_t(buffer_size);
- buffer_b = new int8_t(buffer_size);



optimize kernel - templates

```
templete<unsigned int kernel_size>
void convolution()
{
  for (unsigned y = 0; y < input_height; y++)
  for (unsigned x = 0; x < input_width; x++)
  {
      int sum = 0;

      for (unsigned ky = 0; ky < kernel_size; ky++)
      for (unsigned kx = 0; kx < kernel_size; kx++)
      {
        sum+= kernel[ky][kx]*input[y + ky][x + kx];
      }

      output[y + kernel_size/2][x + kernel_size/2] = (sum*scale)/127;
  }
}

...

convolution<1>(); //1x1 kernel
convolution<3>(); //3x3 kernel
convolution<5>(); //5x5 kernel
```

optimize kernel - unrolling

```
templete<unsigned int kernel_size>
void convolution()
{
  for (unsigned y = 0; y < input_height; y++)
  for (unsigned x = 0; x < input_width; x++)
  {
      int sum = 0;

      if (kernel_size == 3)
      {
        sum+= kernel[0][0]*input[y + 0][x + 0];
        sum+= kernel[0][1]*input[y + 0][x + 1];
        sum+= kernel[0][2]*input[y + 0][x + 2];

        sum+= kernel[1][0]*input[y + 1][x + 0];
        sum+= kernel[1][1]*input[y + 1][x + 1];
        sum+= kernel[1][2]*input[y + 1][x + 2];

        sum+= kernel[2][0]*input[y + 2][x + 0];
        sum+= kernel[2][1]*input[y + 2][x + 1];
        sum+= kernel[2][2]*input[y + 2][x + 2];
      }

      output[y + kernel_size/2][x + kernel_size/2] = (sum*scale)/127;
  }
}
```

optimize kernel - SIMD instructions

elementary row MAC operation in 3x3 convolution :

- **sum**+= **kernel**[0][0]***input**[y + 0][x + 0];
- **sum**+= **kernel**[0][1]***input**[y + 0][x + 1];
- **sum**+= **kernel**[0][2]***input**[y + 0][x + 2];

instruction : SMLADX(int32_t val1, int32_t val2, int32_t val3)

- p1 = **val1**[15:0]***val2**[15:0]
- p2 = **val1**[31:16]***val2**[31:16]
- **res**[31:0] = p1 + p2 + **val3**[31:0]

speed up summary

- input tensor dimensions $16 \times 16 \times 16$
- output tensor dimensions $16 \times 16 \times 16$

convolutional	kernel size	computing time [ms]	speed up ratio
naive	1x1	109.8	1.00
naive	3x3	312	1.00
naive	5x5	532	1.00
template	1x1	26.6	4.13
template	3x3	213.6	1.46
template	5x5	373.3	1.43
template + unrolling	1x1	21	**5.23**
template + unrolling	3x3	57.1	**5.46**
template + unrolling	5x5	106.2	**5.01**

network results

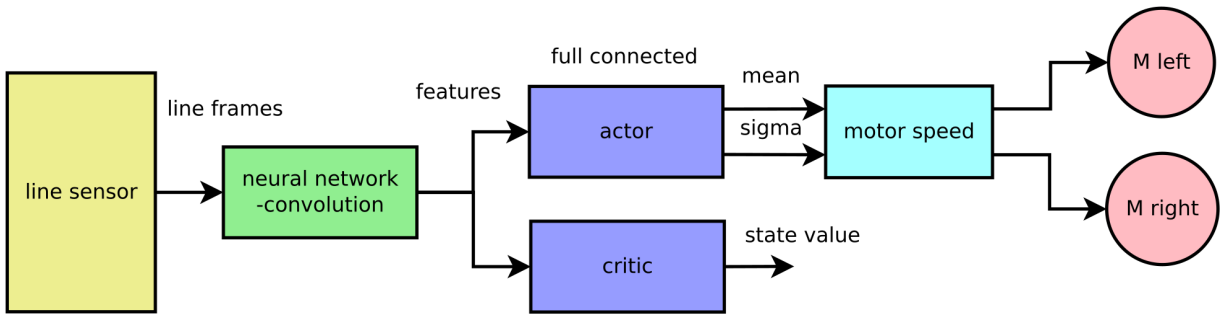
tiny architecture : IN8x8x1 - C3x3x4 - P2x2 - C3x3x8 - P2x2 - FC5
confussion matrix :

1029	12	0	0	0
14	**911**	18	0	0
0	21	**959**	6	0
0	0	21	**968**	18
0	0	0	11	**1012**
98.658	**96.504**	**96.092**	**98.274**	**98.252**

success count 4879
miss count 121
accuracy 97.58%

what is next?

reinforcement learning to take full controll -
Advatage Actor Critic - A2C

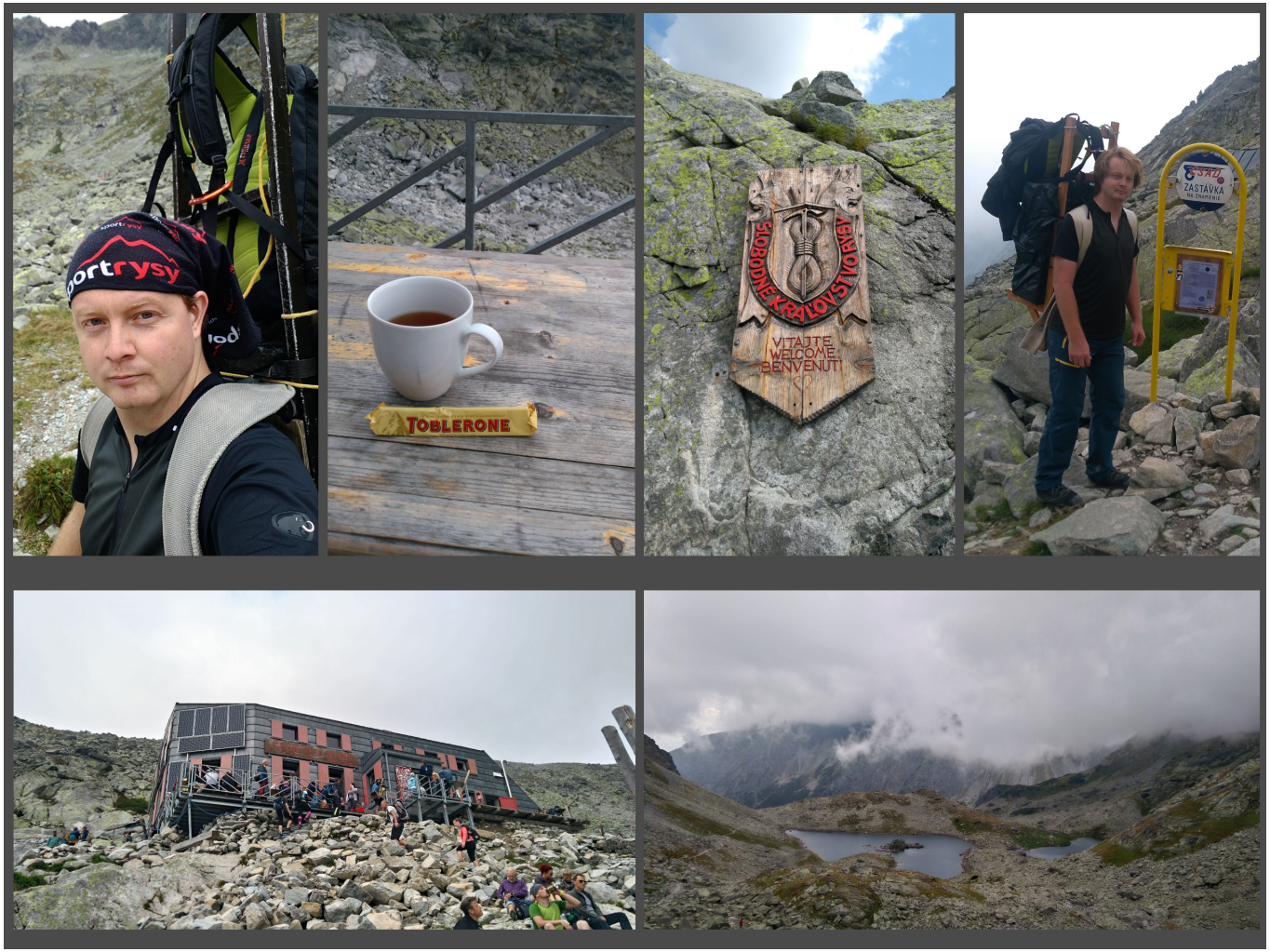


$$\mathcal{L}_{actor} = -\left(R(n) + \gamma Q(s(n+1);\phi) - Q(s(n);\phi)\right) log\pi(s,a;\theta)$$

$$\mathcal{L}_{critic} = \left(R(n) + \gamma Q(s(n+1);\phi) - Q(s(n);\phi)\right)^2$$

$$\mathcal{L}_{entropy} = \sum_a \pi(s,a;\theta) log\pi(s,a;\theta)$$

Q&A



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