Autonomous Robots – Home Work 2

Adam Tonderski, 930524-1037 27/05/2018

1 Kalman Filter

In this task the goal is to implement a Kalman filter to increase the accuracy of position estimations for a differentially steered lawn mower.

1.1 Design

I chose to implement an EKF with the state vector consisting of the position, heading an wheel velocities (not angular velocities). The measurements were position and wheel velocities. No steering signal was used here, since the only natural choice were the wheel velocities and using those felt like cheating. With these conditions, and the equations for a differentially steered robot, the filter takes the following form:

$$\hat{\boldsymbol{x}} = \begin{pmatrix} x \\ y \\ \phi \\ v_L \\ v_R \end{pmatrix} \tag{1}$$

$$f(\hat{\boldsymbol{x}}) = \begin{pmatrix} x + \frac{(v_L+v_R)}{2}\cos(\phi)dt \\ y + \frac{(v_L+v_R)}{2}\sin(\phi)dt \\ \phi - \frac{(v_L-v_R)}{2R}dt \\ v_L \\ v_R \end{pmatrix} \tag{2}$$

$$h(\hat{\boldsymbol{x}}) == \begin{pmatrix} x \\ y \\ v_L \\ v_R \end{pmatrix} \tag{3}$$

From this we can compute the necessary jacobians, $\boldsymbol{F}$ and $\boldsymbol{H}$:

$$\boldsymbol{F}(\hat{\boldsymbol{x}}) = \begin{pmatrix} 1 & 0 & -\sin(\phi)\frac{v_L+v_R}{2}dt & \frac{\cos(\phi)}{2}dt & \frac{\cos(\phi)}{2}dt \\ 0 & 1 & \cos(\phi)\frac{v_L+v_R}{2}dt & \frac{\sin(\phi)}{2}dt & \frac{\sin(\phi)}{2}dt \\ 0 & 0 & 1 & -\frac{dt}{2R} & \frac{dt}{2R} \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{4}$$

$$\boldsymbol{H}(\hat{\boldsymbol{x}}) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{5}$$

We must also define the noise matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$. For simplicity I assumed that they are diagonal, that is the noise in each dimension is completely independent. That isn't necessarily true, but it works decently in this example.

$$\boldsymbol{Q} = \begin{pmatrix} \sigma^2_{pos} & 0 & 0 & 0 & 0 \\ 0 & \sigma^2_{pos} & 0 & 0 & 0 \\ 0 & 0 & \sigma^2_{ang} & 0 & 0 \\ 0 & 0 & 0 & \sigma^2_{vel} & 0 \\ 0 & 0 & 0 & 0 & \sigma^2_{vel} \end{pmatrix} \tag{6}$$

$$\boldsymbol{R} = \begin{pmatrix} \sigma^2_{gps} & 0 & 0 & 0 & 0 \\ 0 & \sigma^2_{gps} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma^2_{odometer} & 0 \\ 0 & 0 & 0 & 0 & \sigma^2_{odometer} \end{pmatrix} \tag{7}$$

These variances are parameters of the filter and good performance depends heavily on setting these to reasonable values, see more about this in the next section. The last component that hasn't been mentioned is the $\boldsymbol{P}$ matrix. It is simply set to identity in the beginning and then updated while the filter is running. Also $\hat{\boldsymbol{x}}$ starts as all zeros, but that can of course be an input parameter to the filter creation.

1.2 Simulation

Our ”true” state was computed separately, outside of the filter. This was done by first generating 2 curves that described the no-slip wheel velocities. Then, these curves were modified with random amounts of slip. Figure 1 shows an example of this. While it might seem like the difference is quite small, the resulting paths are very different. Compare the blue line and the green dashed line in Figure 3 for an example.

The primary difference between the simulated model and the filter's process model is that the filter assumes that the velocities are constant whereas in our simulation the velocities follow a predefined curve (which changes when we add slip into the equation).

1.3 Tuning

In this case it was important to keep the variances in the $\boldsymbol{F}$ matrix quite small, so that the filter would trust its own predictions. After a lot of trial and error I ended up using $\sigma^2_{pos} = \sigma^2_{ang} = \sigma^2_{vel} = 0.00001$. Larger

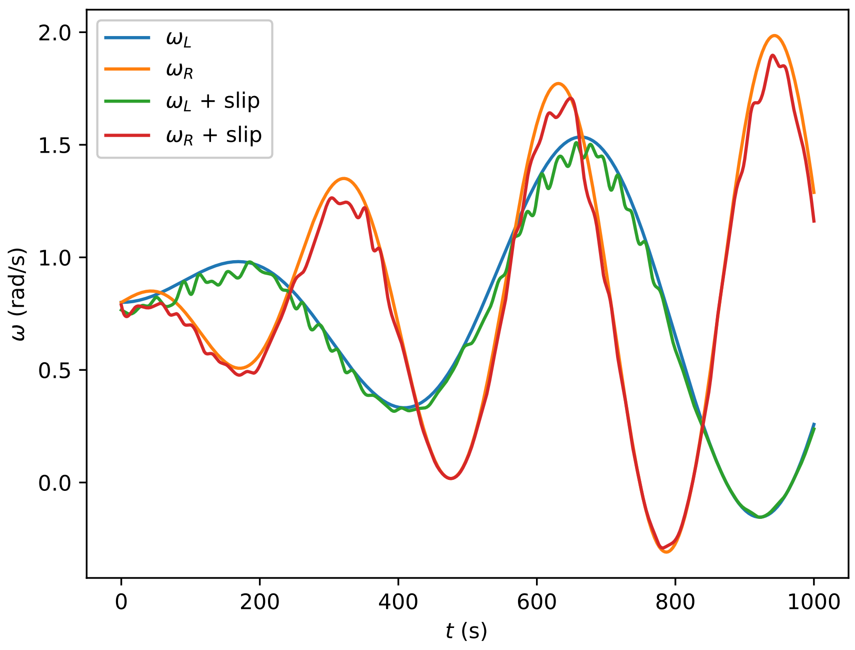

Figure 1: Angular velocity curves for each wheel with and without slip

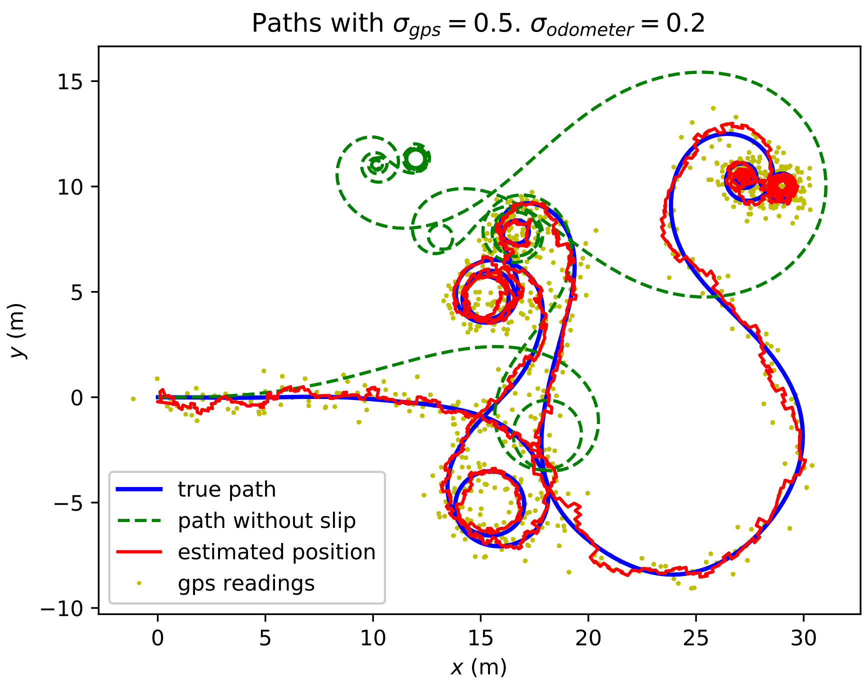

Figure 2: Paths for the same base velocity curve with normal noise. The green dashed corresponds to the original curve, the blue line is the curve after considering slip and the red line is the filter estimation.

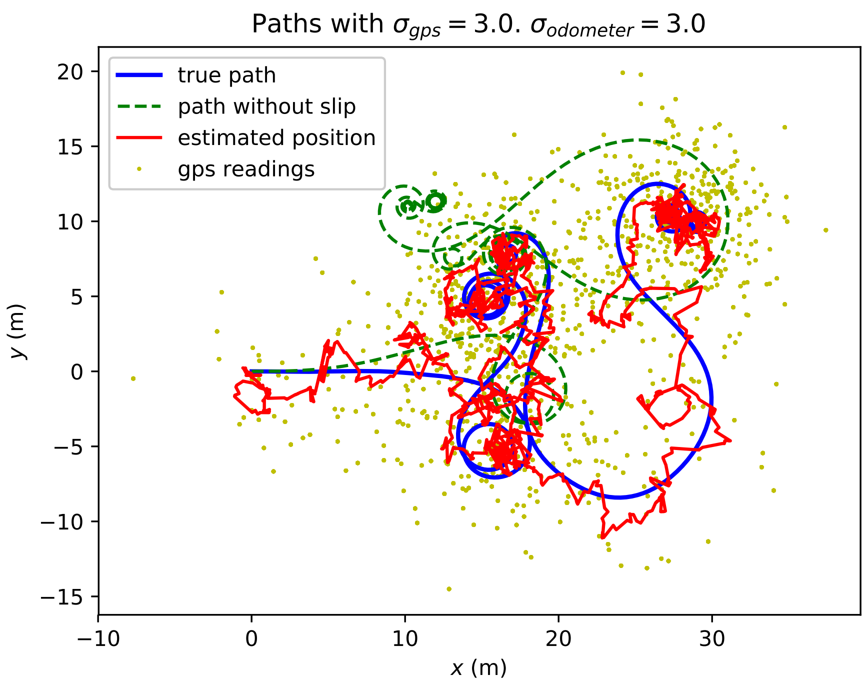

Figure 3: Paths for the same base velocity curve with high noise. The green dashed corresponds to the original curve, the blue line is the curve after considering slip and the red line is the filter estimation.

values caused the error to increase a lot, easily doubling it. This means that our model is very good given that we have the correct state, which makes sense for σ^2_{pos} and σ^2_{ang}, since the only thing we can't account for are changes in velocity. The strange part was that increasing the velocity noise also reduced the filter performance.

σ^2_{gps} and σ^2_{odo} were easier since we could just set them to the same as zero-mean normal distribution that the sensor noise was drawn from. This might seem like inside info, but in reality the sensors would come with a datasheet that contained this information as well.

The final part of the tuning process was trying to deal with the fact that the gps data only gets updated every 20 iterations. This means that it will ”drag us backwards” since the filter has no way of knowing whether the information is fresh or not. This is partly handled by the dynamically updated gain function, but I thought there has to be a way to ”tell” the filter that the information it's getting is outdated. After some reading I found a possible solution. Whenever we pass in an old reading, whether it's from the gps or odometer, we set the corresponding element in $\boldsymbol{R}$ to a very high value (in theory ∞). This way the filter knows that the reading is unreliable and doesn't take it into consideration. The filter was implemented with the ability to toggle between the normal version and the one with the ”noise hack”, so that the performance of both could be evaluated. To distinguish the two versions we will call the original version ”normal” and the other one ”hacky”.

1.4 Results

The metric I used was the average and maximum distance error, which is calculated by simply taking the distance between the estimated and true positions. This error obviously increases as we add more and more noise, so in order to get a feeling for the usefulness of the filter it is compared to the error if we only listened to the gps. The path that was used can be seen in blue in Figure 3. Additionally, estimated positions are shown for the noise values that were given in the assignment. Clearly, the filter works well in this scenario and our estimated position stays close to the true value, even during the sharper turns. It definitely works better than pure gps, the readings of which are marked with yellow dots.

The first result is for comparisons with and without the wheel slip. Using the given values for the sensor noise, the average and max errors were 0.21 and 0.79 without any slip and 0.23 and 0.86 with slip. In other words, the filter performed slightly worse when wheel slip was modeled in the simulation but the difference wasn't very big. The measurements from the gps are enough to course correct and make us follow the correct path.

The results for different amounts of noise, given the same true path and random seed, are presented in Table 1. It turns out that the "hacky" filter outperforms the normal filter under all conditions, which is very interesting to see. As for the effect of noise, it seems that odometer noise is much easier to handle than gps noise, probably due to the different frequencies of the sensors.

It's difficult to construct a good definition for when the filter stops being useful. One could argue that as long as it's doing better than pure gps it works, but normally we would have a spec that tells us how accurate our position estimation has to be. With the first definition, the filter is useful for all levels of tested noise. However, if we require the mean error to be below for example 0.5m, we see that the filter starts failing once the noise goes over 1m for the gps and 1 rad/s for the odometer. By only increasing the noise for one sensor at a time we can also see that the gps is much more sensitive that the odometer. For the former the error increase is almost 1:1, whereas if we only increase the odometer we can handle a standard deviation of 3 rad/s with an acceptable error. One of the reasons for this is probably that much lower frequency of the gps, there simply aren't enough measurements to smooth out the errors. It could also be that our filter places a higher importance on the gps, which makes sense since our model gives perfect position estimations but very faulty velocity estimations (since we assume constant velocity).

Several other paths were tested as well, and the results were very similar for all of them. It would however be interesting to investigate whether one can construct paths that are especially difficult for the filter to handle. For example, rapid, discontinuous, changes in velocity should mess with the filter a lot since our internal model assumes that the velocities are constant. In all of the tested paths the velocity curves had a lot of variations, but the changes were always continuous, which gave the filter time to adapts to the change.

2 EKF Slam

EKF Slam is one of the most traditional approaches to slam. It's basically an EKF where the state additionally consists of landmark positions. A landmark could be any object that we are capable to detect, for example a cone, a wall, a sign, etc. Since it's and EKF, the equations are as follows:

Table 1: Filter performance for varying amounts of noise

gps std-dev	odometer std-dev	hacky mean	hacky max	regular mean	regular max	gps mean	gps max
0	0	0.01	0.05	0.09	0.36	0.09	0.36
0.5	0.2	0.23	0.86	0.33	1.24	0.64	1.97
1	0.2	0.4	1.38	0.52	2.12	1.27	3.79
2	0.2	0.64	2.50	0.85	3.37	2.52	7.42
3	0.2	0.86	3.42	1.21	4.73	3.78	11.05
0.5	0.5	0.27	1.19	0.39	1.38	0.64	1.97
0.5	1	0.31	1.32	0.44	1.54	0.64	1.97
0.5	2	0.34	1.35	0.45	1.47	0.64	1.97
0.5	3	0.38	1.38	0.46	1.66	0.64	1.97
1	1	0.51	2.22	0.7	2.36	1.27	3.79
2	2	1.04	3.61	1.15	3.94	2.52	7.42
3	3	1.66	6.51	1.59	5.58	3.78	11.05

$$\begin{aligned}
\hat{x}_k &= f(\hat{x}_{k-1}, u_k) \\
\hat{P}_k &= F_k P_{k-1} F_k^T + Q_k \\
G_k &= \hat{P}_k H_k^T (H_k \hat{P}_k H_k^T + R)^{-1} \\
x_k &= \hat{x}_k G_k (z_k - h(\hat{x}_k)) \\
P_k &= (I - G_k H_k) \hat{P}_k
\end{aligned}$$

Here, the state vector consists of the regular state variables (for example x, y, ϕ) and also the position of all the landmarks. So the size in this case would be $3 + 2n$, where n is the number of landmarks. f models how the state updates so the first three elements consist of the movement equations (as usual) and are independent of the landmarks. The landmark positions themselves are also assumed to be constant (no moving landmarks). This means that the jacobian F is diagonal, except for upper 3x3 corner, which depends on the motion equations. Since f has the size $3 + 2n$, F has the size $(3 + 2n)x(3 + 2n)$.

As in regular EKF, Q is the process noise covariance matrix, which describes inaccuracies in our model, and it has the same size as F. R is the measurement noise covariance matrix with the size of $(2n)x(2n)$. P_k describes the accuracy of our data and contains all the combinations of covariances between the robot state and the various landmarks. It also has the size $(3+2n)x(3+2n)$. This information is then used to calculate the gain, G_k, which determines (automatically) the balance between listening to the sensors or the internal model estimation. It has the size $(3 + 2n)x(2n)$.

It gets trickier when we get to z h and H. First we need to introduce the concept of range and bearing for a landmark:

$$\begin{pmatrix} r_i \\ \beta_i \end{pmatrix} = \begin{pmatrix} \sqrt{(x - x_i)^2 + (y - y_i)^2} \\ \tan^{-1}(\frac{y_i - y}{x_i - x}) - \phi \end{pmatrix} \tag{8}$$

z_k is the measurment vector which in our case consists of the measured range and bearing for all the known landmarks. Therefore it has the size $2n$. h is a mapping from the state to the measurement data.

In other words, "given the current state, what is our expected measurment". This is exactly the equation above (for a single landmark). So h has the size $2n$. Since H is simply the Jacobian of h, it has the size $2nx(3+2n)$. Note that it is a quite sparse matrix since a landmark observation is doesn't depend on any other landmark observation.

When we detect a new landmark, it is added to the state vector according to the following formula:

$$\begin{pmatrix} x_i \\ y_i \end{pmatrix} = \begin{pmatrix} x_k \\ y_k \end{pmatrix} + \begin{pmatrix} r_i \cos(\beta_i + \phi_k) \\ r_i \sin(\beta_i + \phi_k) \end{pmatrix} \tag{9}$$

This of course increases the sizes of any matrix where the size (as presented above) depends on n, since we now have $n_{new} = n_{old} + 1$.

2.1 Example

Here we will show an example calculation of the matrix H_k. We will make things simple and say that we only have one landmark. Then our state looks like this:

$$\hat{x_k} = \begin{pmatrix} x \\ y \\ \phi \\ x_0 \\ y_0 \end{pmatrix}$$

Using the definitions of range and bearing from before, we get:

$$h(\hat{x_k}) = \begin{pmatrix} r_0 \\ \beta_0 \end{pmatrix} = \begin{pmatrix} \sqrt{(x-x_0)^2 + (y-y_0)^2} \\ \tan^{-1}(\frac{y_0-y}{x_0-x}) - \phi \end{pmatrix} \tag{10}$$

and to get H_k, we simply need to calculate the Jacobian:

$$H_k = \begin{pmatrix} \frac{x-x_0}{r_0} & \frac{y-y_0}{r_0} & 0 & \frac{x_0-x}{r_0} & \frac{y_0-y}{r_0} \\ \frac{\frac{y_0-y}{(x_0-x)^2}}{1+(\frac{y_0-y}{x_0-x})^2} & \frac{\frac{-1}{x_0-x}}{1+(\frac{y_0-y}{x_0-x})^2} & -1 & \frac{\frac{y-y_0}{(x_0-x)^2}}{1+(\frac{y_0-y}{x_0-x})^2} & \frac{\frac{1}{x_0-x}}{1+(\frac{y_0-y}{x_0-x})^2} \end{pmatrix} \tag{11}$$

If we had more measurements H_k would become more and more sparse. For example, if we write the above as:

$$H_k = \begin{pmatrix} H_{s0} & H_{m0} \end{pmatrix} \tag{12}$$

H_k with 3 landmarks would look like this:

$$H_k = \begin{pmatrix} H_{s0} & H_{m0} & 0 & 0 \\ H_{s1} & 0 & H_{m1} & 0 \\ H_{s2} & 0 & 0 & H_{m2} \end{pmatrix} \tag{13}$$

(note that the 0 elements actually correspond to 2x2 blocks with all 0s)

3 Convolutional Neural Networks

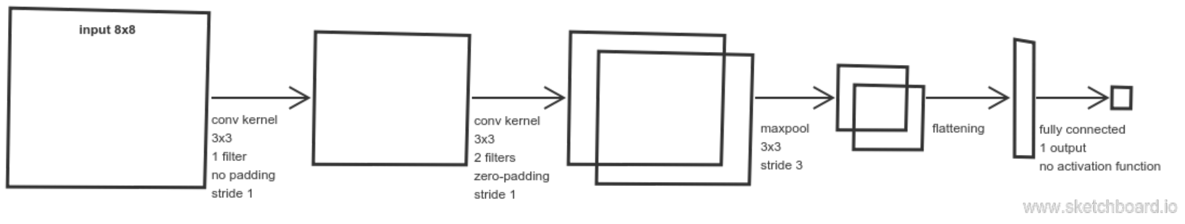

Figure 4: A sketch of the different layers in the network.

A sketch of the network can be seen in figure 4. Given an input, we can manually propagate it through the network and compute the output. Here we will do this for the following input:

$$\boldsymbol{A} = \begin{pmatrix} 3 & 3 & 2 & 1 & 8 & 8 & 9 & 9 \\ 3 & 2 & 2 & 7 & 8 & 9 & 9 & 10 \\ 3 & 3 & 3 & 7 & 7 & 8 & 9 & 10 \\ 3 & 3 & 4 & 6 & 6 & 8 & 8 & 9 \\ 4 & 4 & 5 & 7 & 7 & 8 & 8 & 10 \\ 5 & 5 & 6 & 8 & 8 & 9 & 9 & 10 \\ 5 & 5 & 6 & 8 & 8 & 9 & 9 & 10 \\ 7 & 6 & 7 & 9 & 9 & 9 & 9 & 10 \end{pmatrix} \tag{14}$$

After the first layer we get:

$$\boldsymbol{L1} = \begin{pmatrix} 1 & 4 & -17 & -9 & -12 & -8 \\ -4 & 1 & -12 & -6 & -7 & -10 \\ -1 & -3 & -6 & -2 & -10 & -6 \\ -3 & -4 & -9 & -6 & -8 & -5 \\ -5 & -6 & -11 & -8 & -11 & -9 \\ -3 & -4 & -9 & -6 & -10 & -8 \end{pmatrix} \tag{15}$$

To see how this is calculated, we can show how the first value was calculated. The rest follows the same way and writing it all here would be extremely tedious. Note that during calculation the kernel has to be flipped both vertically and horizontally, otherwise what we are computing is the correlation not convolution.

$$\boldsymbol{K1} = \boldsymbol{K1}_{flipped} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -5 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad \boldsymbol{A}(1:3, 1:3) = \begin{pmatrix} 3 & 3 & 2 \\ 3 & 2 & 2 \\ 3 & 3 & 3 \end{pmatrix} \tag{16}$$

$$\boldsymbol{L1}(1,1) = 0 \cdot 3 + 1 \cdot 3 + 0 \cdot 2 + 1 \cdot 3 + -5 \cdot 2 + 1 \cdot 2 + 0 \cdot 3 + 1 \cdot 3 + 0 \cdot 3 = 1 \tag{17}$$

Then we run through the kernels in layer 2 and end up with two matrices:

$$\boldsymbol{L2_1} = \begin{pmatrix} 1 & -4 & -8 & 17 & 2 & 14 \\ -11 & 30 & 22 & 26 & 14 & 20 \\ -6 & 21 & 14 & 24 & 27 & 16 \\ 0 & 11 & 8 & 31 & 19 & 22 \\ 4 & 15 & 14 & 24 & 14 & 20 \\ 12 & 22 & 16 & 22 & 18 & 0 \end{pmatrix} \tag{18}$$

$$\boldsymbol{L2_2} = \begin{pmatrix} 1.0 & -4.2 & -12.9 & -16.3 & -14.1 & -10.3 \\ -1.5 & -5.9 & -14.4 & -18.0 & -17.3 & -14.1 \\ -3.5 & -7.3 & -11.3 & -13.6 & -14.4 & -12.2 \\ -4.9 & -10.1 & -13.7 & -14.7 & -15.2 & -11.6 \\ -6.6 & -12.5 & -16.6 & -18.1 & -18.1 & -13.5 \\ -4.8 & -9.5 & -12.8 & -14.4 & -15.0 & -11.9 \end{pmatrix} \tag{19}$$

Again, we derive the first value in both matrices. Note that we use zero-padding here, which is why we start the indexing at (0,0) instead of (1,1):

$$\boldsymbol{K2}_{flipped} = \begin{pmatrix} 0 & 0 & -2 \\ 0 & 0 & 0 \\ -2 & 0 & 1 \end{pmatrix} \quad \boldsymbol{K3}_{flipped} = \begin{pmatrix} 0.2 & 0.3 & 0.1 \\ 0.3 & 0.5 & 0.3 \\ 0.1 & 0.2 & 0.1 \end{pmatrix} \quad \boldsymbol{L1}(0:2, 0:2) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 4 \\ 0 & -4 & 1 \end{pmatrix} \tag{20}$$

$$\boldsymbol{L2_1}(1,1) = 0 \cdot 0 + 0 \cdot 0 + -2 \cdot 0 + 0 \cdot 0 + 0 \cdot 1 + 0 \cdot 4 + -2 \cdot 0 + 0 \cdot -4 + 1 \cdot 1 = 1 \tag{21}$$

$$\boldsymbol{L2_2}(1,1) = 0.2 \cdot 0 + 0.3 \cdot 0 + 0.1 \cdot 0 + 0.3 \cdot 0 + 0.5 \cdot 1 + 0.3 \cdot 4 + 0.1 \cdot 0 + 0.2 \cdot -4 + 0.1 \cdot 1 = 1.0 \tag{22}$$

Max-pooling $\boldsymbol{L2_1}$ and $\boldsymbol{L2_2}$ gives us:

$$\boldsymbol{L3_1} = \begin{pmatrix} 30 & 27 \\ 22 & 31 \end{pmatrix} \tag{23}$$

$$\boldsymbol{L3_2} = \begin{pmatrix} 1 & -10.3 \\ -4.8 & -11.6 \end{pmatrix} \tag{24}$$

Where the first and last values (we use 2 to demonstrate the use of a non unity stride) in each matrix were computed according to:

$$\boldsymbol{L3_1}(1,1) = \max(1, -4, -8, -11, 30, 22, -6, 21, 14) = 30 \tag{25}$$

$$\boldsymbol{L3_1}(2,2) = \max(31, 19, 22, 24, 14, 20, 22, 18, 0) = 31 \tag{26}$$

$$\boldsymbol{L3_2}(1,1) = \max(1.0, -4.2, -12.9, -1.5, -5.9, -14.4, -3.5, -7.3, -11.3) = 1.0 \tag{27}$$

$$\boldsymbol{L3_2}(2,2) = \max(-14.7, -15.2, -11.6, -18.1, -18.1, -13.5, -14.4, -15.0, -11.9) = -11.6 \tag{28}$$

Finally, we flatten $\boldsymbol{L3_1}$ and $\boldsymbol{L3_2}$ and connect the resulting vector to the single output neuron, according to the standard formula $y = \boldsymbol{w}^T\boldsymbol{x}$:

$$y = \begin{pmatrix} 0.1 & 0.2 & 0.2 & 0.1 & 0.2 & 0.1 & 0.1 & 0.1 \end{pmatrix} \begin{pmatrix} 30 \\ 27 \\ 22 \\ 31 \\ 1 \\ -10.3 \\ -4.8 \\ -11.6 \end{pmatrix} = 13.43 \tag{29}$$

This means that the output of the entire network is 13.43, unless of course there were computational errors.

4 Essay

Most people would agree that it’s no longer a matter of if autonomous systems are going to get integrated into every part of our lives but rather when. In this brief opinion piece, I will discuss 3 of the, in my opinion, most relevant areas: transportation, healthcare and industry. Afterwards I will try to address some of the potential problems related to unemployment and inequality. Due to the limited nature of this essay I chose ignore many areas entirely, for example the military would have warranted an entire essay on its own.

4.1 Transportation

Transportation is probably getting the most media attention when it comes to automation. There is a huge amount of money flowing into the development of autonomous vehicles. While a lot of discussion is around personal vehicles, I personally think that it is the least interesting piece of the puzzle. Instead of talking about ”ifs” or ”whens”, I think we should ask ourselves ”why”. Regular people owning a self-driving car that just sits around most of the time doesn’t really make any sense. Instead, the true revolution lies in smarter and more efficient shared transportation built on a fleet of vehicles of varying sizes and capabilities. We already see the hints of this with car-pooling services such as Uber Pool and Lyft Line.

My hope is that my generation never has to own a car, but rather use a combination of a more frequent and reliable public transportation and point-to-point pooling services. Occasionally we will have to rent a private vehicle for longer trips to unfrequented locations. The benefit of such a system is enormous. First of all, it reduces total amount of vehicles a lot. Secondly, with sufficiently smart algorithms, the number of vehicles on road should reduce as well, if everyone can share rides efficiently. This in turn reduces the environmental impact of the transportation industry, which today is one of the main contributors to pollution and global warming.

This beautiful utopian picture won’t happend overnight, and is probably one of the later developments when it comes to transportation. A much more immediate use-case are self driving trucks. Smart self-driving truck convoys are likely to start appearing on the roads in the next 10 years, and once the magic cost-efficiency breaking point is reached, they will probably spread really fast. There are millions of truck drivers in the US alone so there is clearly a lot of money to be saved here. This development will probably not be very noticeable to the average person though, since the majority of time for a truck is spent on the highways. Most of the impact will probably come from the political discussions that will emerge from the job displacement, but more on that in a later section.

4.2 Healthcare and industry

As the population living longer and longer, the health care cost per person is skyrocketing. This is a global phenomenon that is likely to be one of the major challenges of this century. While breakthroughs in medicine that let people stay healthy longer are probably necessary, there is also a lot of potential gain to be found from deploying advanced, automated systems.

A very iconic application, that is currently explored in Japan, are partner robots for the elderly, also called carebots. The idea here is not just to provide physical assistance 24/7, but also to be a conversation partner and provide stimulation. There is a long way to go before you can play tennis with your own robot partner, but early more specialized prototypes exist today. It is not at all a stretch to expect these kinds of systems to be deployed on a small scale in 10-15 years. Sweden will probably follow a bit after Japan, but hopefully we can see some use of carebots in at least 20 years. The idea here is not just to avoid a societal collapse under the increasing tax burden to sustain an old population. Rather, it is to improve the health and happiness of elderly people, while at the same time reducing the labour requirements.

There are many many more specialized applications of autonomous systems in healthcare. For example mechanical surgery, which far exceeds the precision of humans, medicine delivery, early detection by continuous monitoring, etc.. There are also tons of very specific micro-manage problems that doctors and nurses have to think about that could easily be automated. One such example I recently learned about is the fact that patients tend to develop internal bleeding if the lie in the same position for long periods of time. It would be straightforward to develop a bed that can detect this, and adjust the shape slightly to alleviate pressure from the affected areas. The social benefits are clear: improved quality by giving the doctors and nurses more time to do what they are best at. Some of these things are already in use and more and more will roll out in the next decades. It's definitely a matter of when and not if.

Finally, due to the unusually high quality requirements, the operation and maintenance of hospitals is extremely complex and expensive. This can be simplified by converting the entire building to an automated system which monitors and optimizes itself. For example if the air quality drops below the desired quality the overseeing system will immediately detect that fact, attempt to repair itself, and if that doesn't work it can notify someone who can look at it manually.

This idea of autonomous building is actually called Industry 4.0 and is gaining a lot of traction. The great power in this approach is in the ability to deploy sophisticated, general, algorithms that use the sensor data and to learn to optimize the efficiency of the factory (or whatever it is). For example, Google Deepmind was able to use a general purpose algorithm to reduce the cooling cost for a data centre by up to 40%.

The biggest short term societal impact of the Industry 4.0 ”revolution” will most likely be the loss of jobs, much as in the case of trucks. It should of course also bring higher quality and/or lower prices, but those are of less interest here. The potential problems and solutions associated with the job displacement associated with automation will be discussed in the next section.

4.3 Inequality and unemployment

According to a recent McKinsey study, around half of the current jobs can be automated using technology that has been demonstrated today. While there is a big difference between demonstration and production, it still shows that we are in the beginnings of a huge shift in the workforce. The big question is of course what will happen to all the people that are about to lose their jobs in coming decades?

During previous revolutions, this issue has mostly resolved itself since the workforce was able to move from the primary and secondary sectors (acquiring raw materials and manufacturing) to the tertiary sector (services). While it's possible that new jobs (or even sectors) will be created faster than the old ones are automated, we must think about the possibility that they might not. After all we are on the brink of having the ability to create specialized systems that can outperform humans in almost any narrow area. Creativity and extreme adaptability are our biggest strengths, but they are extremely underutilized in most jobs.

Another aspect that is different this time is that the power and wealth of a company/institution is no longer proportional to the number of employees. In the ”old/current system” there was an implied mutual codependency which gave the workers a fair amount of power (for example in the form of strikes). In today’s globalized world that is less and less true, especially with automation taken into account. Unless some measures are taken this could further increase the already growing income (and power) inequalities.

Here is a very dystopian thought example. The fall of communism in Europe is attributed in great part to the massive peaceful strikes that began in Poland. The workers were threatened with violence but bunkered up in the factories and simply refused to work. This strategy forced concessions from the government and was a wild success that spread to many other countries. Imagine now a world in which all the mines and factories were completely automated. No workers, no strikes, no concessions. I’m not saying communism would still rule in eastern Europe, but things would definitely have been different!

A couple solutions to these potential problems have been gaining popularity in the recent years. The ones I will discuss here are universal basic income and reduced working hours. Both of them are based on the idea that instead of finding more work for everyone we can just let everyone work less.

In the case of universal basic income, everyone gets a fixed paycheck each month. Any employment would of course add on top of that. Reduced work weeks are a bit simpler as it means that instead of 1 person working for 8 hours we could have 2 people working 4 hours each. In this example we have doubled the number of employed people without creating any extra work.

The problem with both of these ideas is that the money has to come from somewhere. There have been small-scale trials which worked very well but in these cases the money came from taxes that are mostly payed by other people that work. In order for these ideas to work, there needs to be a fair and sustainable way to distribute money, even if only a minority of the population is working. Some suggestions involve paying taxes for automated systems as if they were workers, and others suggest complete reworks of the taxation system (such as removing income tax entirely).

All in all I think these are societal and political challenges that we need to take seriously sooner rather than later since in only 30 years almost half of our current jobs could be gone.

4.4 Conclusion

I am very much looking forward to seeing what the next decades years have in store for us. I strongly believe that in the long term the prevalence of automated systems is a good thing for society since it lets us humans focus on what we are good at rather than performing repetitive, mechanical tasks. Besides, most of these systems are not just as good as humans, they vastly outperform us. This can lead to either improved quality of products/services or reduces prices. Hopefully both!

However, the transition period could be very tumultuous, especially in industries that are highly exposed to automation. Once a critical cost efficiency is reached unemployment in those fields will rise extremely fast, and some social net will probably have to be in place to help the affected people in their transition.