TTK4250 Sensor Fusion
Assignment 4

Hand in: *Friday 4. October 23:59* on Blackboard.

Read Python setup guide on BB, it can be found under *Course work*.
This assignment should be handed in on Blackboard, as a PDF pluss **the zip created by running *create_handin.py***, before the deadline.
You are supposed to show how you got to each answer unless told otherwise.
If you struggle, we encourage you to ask for help from a classmate or come to the exercise class on Friday.

Task 1: *Gaussian mixture reduction*

(a) Finish the `gaussian_mixture.GaussianMixture.mean` method.

(b) Finish the `gaussian_mixture.GaussianMixture.cov` method.

(c) You are waiting for a friend who is driving to you on a particular route. While you are waiting, you deduce a distribution over how far you believe he has come from a couple of messages from him. The messages were a bit unclear, so you ended up with a univariate Gaussian mixture with three components as the distribution. However, you think it is a bit much with three components and believe that you should get a good enough representation with only two Gaussians. So, you decide to merge two of them by making these two into a new Gaussian that matches the mean and covariance of the mixture of the two. To keep it simple you neglect any time/speed aspect.

In the given Gaussian mixtures, which would you merge by moment matching if you were to merge two components, why would you merge these, and what would the resulting components be?

There is not necessarily a one true answer here, and it depends on what you emphasize.

Hint: Section 6.3 in the book can provide some insight. For merging you can use the fact that $w_1 p_1(x) + w_2 p_2(x) + w_3 p_3(x) = w_1 p_1(x) + (w_2 + w_3)\left(\frac{w_2}{w_2+w_3} p_2(x) + \frac{w_3}{w_2+w_3} p_3(x)\right)$.

When you run the `main.py` file you will see a plot of the mixtures, based on the methods implemented in a) and b).

i. $w = \{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\}$, $\mu = \{0, 2, 4.5\}$ and $P = \{1^2, 1^2, 1^2\}$.
ii. $w = \{\frac{1}{6}, \frac{4}{6}, \frac{1}{6}\}$, $\mu = \{0, 2, 4.5\}$ and $P = \{1^2, 1^2, 1^2\}$.
iii. $w = \{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\}$, $\mu = \{0, 2, 4.5\}$ and $P = \{1^2, 1.5^2, 1.5^2\}$.
iv. $w = \{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\}$, $\mu = \{0, 0, 2.5\}$ and $P = \{1^2, 1.5^2, 1.5^2\}$.

Task 2: *Measurement likelihood of the interacting multiple model filter and particle filter*

(a) Derive the measurement likelihood, $p(z_k|z_{1:k-1})$, of an IMM filter by using of the total probability theorem,

$$p(z_k|z_{1:k-1}) = \sum_{s_k} \int p(z_k|x_k, s_k) p(x_k|s_k, z_{1:k-1}) \Pr(s_k|z_{1:k-1}) \, \mathrm{d}x_k,$$

in terms of the mode likelihood, $\Lambda_k^{s_k}$, and mode probabilities, $\Pr(s_k|z_{1:k-1})$, assuming $p(z_k|x_k, s_k) = p(z_k|x_k)$.

> *Hint:* Use terms and equations from section 6.2.

(b) Assume that the PF state distribution is $p(x_k|z_{1:k-1}) \approx \sum_{i=1}^{N} w_k^i \delta(x_k - x_k^i)$, where $\delta(x)$ is the Dirac delta function, w_k^i are the normalized weights and x_k^i are the particles. Derive the measurement likelihood, $p(z_k|z_{1:k-1})$, of this PF.

> *Hint:* You can use the total probability theorem to be able to use the given approximate PF state distribution.

Task 3: *Implement an IMM class*

Finish implementing the IMM filter. You have to implement steps 1 through 4 of the IMM algorithm as given in the book, in addition to a 5th estimation step.

(a) Finish the `filter.FilterIMM.calculate_mixings` method.

(b) Finish the `filter.FilterIMM.mixing` method.

(c) Finish the `filter.FilterIMM.mode_match_filter` method.

(d) Finish the `filter.FilterIMM.update_probabilities` method.

(e) Finish the `filter.FilterIMM.step` method.

Task 4: *Tune an IMM*

This last task is very open-ended. All the parameters you are intended to change are located in `tuning.py`. Note that some parameters might cause the system to crash, due to the naive implementation and numerical error.

As there is a lot of new code, and no guide on how to use it, it's understandable if this task is difficult. It is okay to skip it long as you have given an honest attempt at trying out different parameters.

(a) Try out different noise values for the dynamic model and sensor model used in the simulation. Can you come up with a hypothesis on when the filter has a hard time correctly estimating the modes?

> *Hint:* Set the seed to 'None' to generate random paths on every run.

(b) Currently, the filter knows the optimal parameters, as it is using the same dynamic model and sensor model as the simulation. Try to use different dynamic and/or sensor models for the filter. You can for example change the noise parameters of the sensor or the hold times of the dynamic model. How does this affect the estimates and the NIS and NEES values?