Human and animal behavior classification through DKFs

Samuel Bortolin, Alessandro Grassi

***Abstract*—The aim of the project is to simulate inside Matlab a system able to perform the classification between *human* and *animal* class of an entity that moves through the environment. The classification is based on the assumption that animals can move in a random manner while humans move in an almost constant direction. The data is acquired in a distributed way through a network of sensors and shared among neighbors via the consensus algorithm. Each sensor estimates the position of the entity using two distinct Kalman filters, one for each class. The class associated with the highest probability of fitting the measurements over time with respect to the pdfs outputted by the two Kalman filters will be the one predicted by our distributed system. Simulation results demonstrate that the developed solution is very effective in this classification task reaching the peak performance of 99.2% of accuracy in ideal conditions and with the hyperparameters of our algorithm properly tuned.**

***Index Terms*—Wireless sensor networks, distributed Kalman filtering, consensus algorithm, entity classification.**

I. Introduction

Kalman filters are a powerful tool that can be used in many different applications. Essentially, they are a technique that allows combining prior data on an entity model with novel data coming from a sensor. The output is a posterior pdf of the entity position estimate obtained starting from a prediction step using the entity model and successfully integrating the information coming from the sensors in an update step. Since the entity under measure is considered as moving and the fact that the estimate comes from a probability distribution make Kalman Filters very suitable for trajectory estimation. It is for this reason that WLS can not be used for such a purpose, in fact WLS assumes the entity to be estimated as a fixed unknown variable. The Kalman filters' versatility can be shown in this work where we use them not only to track the entity, but also to classify it.

Distributed estimation allows us to obtain better results thanks to the fact that many different sensors in different positions can share their knowledge in order to reach a consensus with the others. In this way eventual noise collected can be attenuated if not canceled. The benefit of having nodes that share knowledge with each other instead of having a centralized node that collects and elaborates information is an increased fault tolerance of the network.

The tracking of multiple models is a well studied problem and in [1] Bar-Shalom et al. presented some tracking strategies involving multiple models. Of particular interest for us is the static multiple model estimator that uses a Kalman filter for each model for obtaining mode-conditional estimates and covariances. Using this technique it is possible to obtain and update at each iteration of the Kalman filter the mode probabilities.

In [2] Li & Jia presented a technique that allows them to use the consensus algorithm for solving a distributed estimation problem involving multiple models. They want to track an entity in a distributed manner and simultaneously detect the model of the entity that could change over time. In developing our solution we take inspiration from the algorithm they presented in the paper.

A. *Problem Formulation*

The problem we are trying to solve is the classification of an entity that moves through an environment that is assumed to be not urban, e.g., a forest. The discrimination happens between two different classes: *human* and *animal*, and will be performed through the analysis of the movement of such entities assuming that animals do not follow any specific trajectory, so they can move in a random way, and humans move keeping an almost constant direction and velocity. The data is collected by a network of sensors that every timestep estimates the position of the entity. Each sensor has a known covariance matrix and has a probability of failing when performing the detection of the entity, this probability is considered since we want to simulate that it may happen that even if the entity is inside the range of the sensor it may fail to detect the entity. Each sensor can communicate with others in order to reach a consensus about the position of the entity to be estimated, the range of communication is limited and we consider a chance that the communication may fail.

II. Adopted Models

A. *Communication System*

As a distributed system we used a network of sensors. Sensors are static and randomly placed in the environment, but they know their own position since they are provided with a GPS. The communication between sensors is wireless and happens through a Bluetooth Low Energy (i.e., Bluetooth LE) since it allows a good communication range and allows a sufficient quantity of data transmitted but with a low energy consumption. The sensors are randomly placed in a square field of 1 km per side. Each sensor has a limited range of communication of about $500\ meters$, it is not possible to communicate with the sensors above this distance. We assumed also a probability of the 0.1% that the sensor fails to communicate with other sensors even if they are in the range.

B. *System Model*

Each sensor is equipped with a stereo 360º thermal camera and it is able to detect the entity and determine the position of an entity with negligible bias and a standard deviation of $25\ meters$ for both the X and the Y axis. Its model can be described like so:

$$Z\ = p\_entity\ + N(0,\ sigma\_z\_value\ *\ I)$$

where Z is the *2x1* vector representing the estimated position of the entity, each value contains respectively the X and Y value of the cartesian axis. p_entity is the vector containing the actual cartesian coordinates of the entity position. N is the normal distribution, its mean is 0 (a *2x1* vector containing 0) and its standard deviation is $sigma\_z\_value$.

Each sensor is assumed to be equipped with a GPS so they are able to know their own position with respect to the global frame. The error due to the estimation of the position of the GPS can be disregarded since it is assumed to be negligible with respect to the variance of the computational process that allows to get the distance and direction of the entity from the sensor. The sensors are able to estimate the global position of the entity (with negligible bias) since they know its own position (with negligible variance) and they are able to compute the relative distance and direction of the entity. In fact, a processing phase using a neural network is done on thermal camera images and it is able to provide as output the estimated relative distance and direction of the entity with respect to the position of the sensor. Each sensor has a limited range of detection of about $250\ meters$, above this threshold, it is not possible to detect the entity and compute its position. We assumed also a probability of the 0.1% that the sensor fails to detect the entity even if it is in the range. The way in which the measurement is obtained can be described like so:

$$Z\ = p\_sensor\ +\ B\_entity\ *\ d\_entity$$

where Z is the *2x1* vector representing the measured distance, each value contains respectively the X and Y value of the cartesian axis. p_sensor is the vector containing the cartesian coordinates of the sensor position. d_entity and B_entity represent the estimated relative distance and direction of the entity.

III. Solution

As anticipated in the previous sections, we performed the classification by analyzing the trajectory followed by the entity. In particular one Kalman filter for each class has been used and the Kalman filter that fits better the measurements over time determines the predicted class. The two classes are *human* and *animal*. In developing our solution we take inspiration from [3] where Olfati-Saber introduced some novel distributed Kalman filtering (DKF) algorithms for sensor networks.

The prediction step of the Kalman filter that uses the prior knowledge given by the dynamics of the *human* model is defined as follows:

$$p\_human\_pred(k) = A * p\_human(k-1) + \\ + direction\_human\_predicted * u\_human\_predicted$$

$$P\_human\_pred(k) = A * P\_human(k-1) * A^t + sigma\_noise\_human^2 * I$$

where A defines the system matrix of the human model, in our case is just the identity matrix, $p\_human(k-1)$ and $P\_human(k-1)$ are the previous mean vector and covariance matrix of the entity position estimate. $direction\_human\_predicted$ is the direction column vector composed by the estimated directions: the abscissa and the ordinate. Such a vector is estimated by computing the vectorial difference between two distinct measurements and getting its polar coordinates after a certain number of iterations (at least $5\,iterations$ if we have a measurement, if not we wait until we get one). $u\_human\_predicted$ represents the norm of the step that the entity takes, it is computed by taking the norm of the difference vector just described before divided by the number of iterations considered. Before reaching the iterations needed for computing these terms they are simply not used in the previous formulas. Once computed, these values are then averaged over all the sensors that collected measurements using the consensus algorithm for $21\,iterations$, this value is empirically found as a good tradeoff between convergence and energy consumption for the communication. The consensus parameters are computed at node startup using the Metropolis-Hastings weights. $sigma\_noise\_human$ is the standard deviation of the human entity noise that affects the position update of the human entities and this noise independent from the measurement noise. To be noticed that in order to preserve the coherence of the Kalman filter, the estimated values coming from previous measurements are used starting from the next iteration of the algorithm.

The prediction step of the Kalman filter for the *animal* class is done instead as follows:

$$p\_animal\_pred(k) = A * p\_animal(k-1) + \\ + (direction\_animal\_predicted * u\_animal\_predicted) / scaling\_factor$$

$$P\_animal\_pred(k) = A * P\_animal(k-1) * A^t + sigma\_noise\_animal^2 * I$$

where A defines the system matrix of the animal model, as before just an identity matrix, $p\_animal(k-1)$ and $P\_animal(k-1)$ are the previous mean vector and covariance matrix of the entity position estimate. For the animal model we cannot say anything a priori about the control input since it is assumed randomly changing both in direction and intensity. We just assume that the entity takes a step proportional to the intensity of the previous one and in the same direction. $direction\_animal\_predicted$ is the direction column vector composed by the estimated directions: the abscissa and the ordinate. Such a vector is estimated by computing the vectorial difference between the two previous measurements and getting its polar coordinates. $u\_animal\_predicted$ represents the norm of the step that the entity takes, it is computed by taking the norm of the difference vector just described before. The resulting vector is divided by a $scaling\_factor$ in order to not give a lot of confidence to the previous step. If the previous two measurements are not present, these values are simply not used in the previous formulas. Once computed, these values are then averaged over all the sensors that collected measurements using the consensus algorithm. $sigma\_noise\_animal$ is the standard deviation of the animal entity noise that affects the position update of the animal entities and this noise independent from the measurement noise. To be noticed that in order to preserve the coherence of the Kalman filter, the estimated values coming from previous measurements are used starting from the next iteration of the algorithm.

After the prediction step follows the update step, since we deal with a network of sensors we developed a distributed Kalman filter algorithm that enables the sharing of sensor data among nodes. Each node that has successfully collected a measurement of the entity initializes the vector a_i and the matrix F_i as: $a_i = H_i * R_i^{-1} * z_i$ and $F_i = H_i^t * R_i^{-1} * H_i$ where R_i

is the covariance matrix of the sensor noises and then the consensus algorithm is performed with such values in order to reach an agreement. To be notice that in our specific case the sensor matrix H_i is assumed to be an identity matrix for each sensor. During the consensus procedure each node also computes the number of nodes that have a measurement at the current time step. This is done by initializing each node with the value *node_estimate* equal to 1 for each node having a measurement and then sharing it in a dedicated consensus cycle with all the others set to 0, then the estimated number of nodes can then be retrieved by computing the inverse of *node_estimate*. Since there is the possibility of communication or entity detection failure it can happen that some nodes do not have any measurement. To solve this issue each node that has a measurement shares the *a, F* and *node_estimate* to its neighbors that don't possess any measurement. This process is repeated for more steps to ensure that every node has *a, F* and *node_estimate*. Since this share is performed after the consensus algorithm every node receives the same values. Then the update step is completed as follows:

$$P\_human(k) = (P\_human\_pred(k)^{-1} + node\_estimate(k) * F(k))^{-1}$$

$$p\_human(k) = p\_human\_pred(k) + P\_human\_pred(k) * node\_estimate(k) *$$
$$* (I - node\_estimate(k) * F(k) * P\_human(k)) * (a(k) - F(k) * p\_human\_pred(k))$$

$$P\_animal(k) = (P\_animal\_pred(k)^{-1} + node\_estimate(k) * F(k))^{-1}$$

$$p\_animal(k) = p\_animal\_pred(k) + P\_animal\_pred(k) * node\_estimate(k) *$$
$$* (I - node\_estimate(k) * F(k) * P\_animal(k)) * (a(k) - F(k) * p\_animal\_pred(k))$$

We derived these expressions from the centralized Kalman filter update step:

$$p(k) = p\_pred(k) + P\_pred(k) * H^t * (H * P\_pred(k) * H^t + R)^{-1} * (z(k) - H * p\_pred(k))$$

$$P(k) = P\_pred(k) - P\_pred(k) * H^t * (H * P\_pred(k) * H^t + R)^{-1} * H * P\_pred(k)$$

Using the matrix inversion lemma:

$$(A + B * C * B^t)^{-1} = A^{-1} - A^{-1} * B * (B^t * A^{-1} * B + C^{-1})^{-1} * B^t * A^{-1}$$

with the following substitutions $A^{-1} = P\_pred(k)$, $B^t = H$ and $C^{-1} = R$ we obtained the following expression for the covariance matrix:

$$P(k) = (P\_pred(k)^{-1} + H^t * R^{-1} * H)^{-1}$$

where we substituted $H^t * R^{-1} * H$ with $node\_estimate(k) * F(k)$ after reaching the consensus on both the quantities obtaining the final expression:

$$P(k) = (P\_pred(k)^{-1} + node\_estimate(k) * F(k))^{-1}$$

For the mean vector we use the matrix inversion lemma with the following substitutions $A = R$, $B = H$ and $C = P\_pred(k)$ obtaining:

$$p(k) = p\_pred(k) + P\_pred(k) * H^t *$$
$$* [R^{-1} - R^{-1} * H * (H^t * R^{-1} * H + P\_pred(k)^{-1})^{-1} * H^t * R^{-1}] * (z(k) - H * p\_pred(k))$$

We can notice that we have obtained $(H^t * R^{-1} * H + P\_pred(k)^{-1})^{-1} = P(k)$ and so we can do the substitution:

$$p(k) = p\_pred(k) + P\_pred(k) * H^t *$$
$$* (R^{-1} - R^{-1} * H * P(k) * H^t * R^{-1}) * (z(k) - H * p\_pred(k))$$

Collecting $H^t * R^{-1}$ from the first parenthesis and moving it to the last one we got:

$$p(k) = p\_pred(k) + P\_pred(k) *$$
$$* (I - H^t * R^{-1} * H * P(k)) * (H^t * R^{-1} * z(k) - H^t * R^{-1} * H * p\_pred(k))$$

where we substituted $H^t * R^{-1} * H$ with $node\_estimate(k) * F(k)$ and $H^t * R^{-1} * z(k)$ with $node\_estimate(k) * a(k)$ after reaching the consensus on all the quantities obtaining the final expression:

$$p(k) = p\_pred(k) + P\_pred(k) * (I - node\_estimate(k) * F(k) * P(k)) *$$
$$* (node\_estimate(k) * a(k) - node\_estimate(k) * F(k) * p\_pred(k))$$

Collection $node\_estimate(k)$ from the last parenthesis we got:

$$p(k) = p\_pred(k) + P\_pred(k) * node\_estimate(k) *$$
$$* (I - node\_estimate(k) * F(k) * P(k)) * (a(k) - F(k) * p\_pred(k))$$

After the update step we update the probability of models if we reached the minimum number of iteration for computing the estimated direction and the norm of the step for the human prediction step. If the nodes has a measurement the probabilities for the 2 modes are computed as follows:

$$\Lambda\_human(k) = prob(\, z(k) \mid model = human,\, z(1), \ldots, z(k-1)\,)$$

$$\Lambda\_animal(k) = prob(\, z(k) \mid model = animal,\, z(1), \ldots, z(k-1)\,)$$

$$prob\_human(k) = \frac{\Lambda\_human(k) * prob\_human(k-1)}{\Lambda\_human(k) * prob\_human(k-1) + \Lambda\_animal(k) * prob\_animal(k-1)}$$

$$prob\_animal(k) = \frac{\Lambda\_animal(k) * prob\_animal(k-1)}{\Lambda\_human(k) * prob\_human(k-1) + \Lambda\_animal(k) * prob\_animal(k-1)}$$

IV. Implementation Details

In order to simulate the movement of the entity in Matlab we generate a random set of coordinates from which the entity will start, the coordinates are sampled with a uniform probability around the center of the field where the sensors are placed. Then we generate a random direction that will be followed. In the case of the human, the direction remains unchanged for all the simulation, instead for the animal the direction randomly changes with a certain probability between one iteration and the other. The simulation of the entity also includes some noise on the length of the step, to do that we randomly lower or raise the norm of the step by a certain value depending on the class of the entity. This starts from a value of $20\, meters$ and varies with uniform probability in range $[-\,2.5,\, 2.5]\, meters$ for the animal model and in range $[-\,0.05,\, 0.05]\, meters$ for the human model. We also add noise to the next step position, these are the two hyperparameters coming from the human and animal model. The entity position update can be described by the following formulas:

$$p\_entity\_human(k) = p\_entity\_human(k-1) + \\ + direction\_fixed * u(k) + N(0,\, sigma\_noise\_human * I)$$

$$p\_entity\_animal(k) = p\_entity\_animal(k-1) + \\ + direction\_randomized(k) * u(k) + N(0,\, sigma\_noise\_animal * I)$$

where:

$$u(0) = 20\, meters$$

$$u(k) = u(k-1) + U(-\,0.05,\, 0.05) \quad \text{for human entities}$$

$$u(k) = u(k-1) + U(-\,2.5,\, 2.5) \quad \text{for animal entities}$$

where U is the uniform distribution.

For what concerns the Kalman filter model we initialize the mean vector of the prior to the center of the field and its covariance matrix to the identity matrix multiplied by 10^6, this is true for each sensor and each model.

The update for the probability of modes is implemented in Matlab as follows using the mvnpdf (multivariate normal pdf) function:

$$\Lambda\_human(k) = mvnpdf(\, z(k),\, p\_human(k),\, P\_human(k)\,)$$

$$\Lambda\_animal(k) = mvnpdf(\, z(k),\, p\_animal(k),\, P\_animal(k)\,)$$

$$prob\_human(k) = \frac{\Lambda\_human(k) * prob\_human(k-1)}{\Lambda\_human(k) * prob\_human(k-1) + \Lambda\_animal(k) * prob\_animal(k-1)}$$

$$prob\_animal(k) = \frac{\Lambda\_animal(k) * prob\_animal(k-1)}{\Lambda\_human(k) * prob\_human(k-1) + \Lambda\_animal(k) * prob\_animal(k-1)}$$

Due to some rare problems with the *mvnpdf* function of Matlab with the covariance matrices, in which it raises an error saying that they must be symmetric and positive definite, we introduce a regularization term that starts from 0 and iteratively increases by 1 when needed:

$$\Lambda\_human(k) = mvnpdf(\, z(k),\, p\_human(k),\, P\_human(k) + regularization * I\,)$$

$$\Lambda\_animal(k) = mvnpdf(\, z(k),\, p\_animal(k),\, P\_animal(k) + regularization * I\,)$$

In order to test the performance of the proposed solution, we wrote a script that performs several simulations and collects the results. First of all, we turned the entire main code into a Matlab function and moved all hyperparameters as input of this function. Then we created a script that first performs *n* experiments simulating human behavior and then *n* experiments simulating animal behavior. For each run, we take note of every *true negative, true positive, false negative* and *false positive*, in this way we can build up the confusion matrix. We also take note of the disposition of the sensors (that is random for each experiment), the mean error of the predictions for both models and also for the combination of the two, and the confidence of the prediction. The default configurations of the hyperparameters that we used is the following:

- The number of sensors/nodes of the network: $node\_number = 50$;
- Nodes can communicate with other nodes up to a certain distance: $max\_communication\_range = 500\, meters$;
- Probability of failure to communicate: $communication\_failure\_prob = 0.001$;

- Sensors noise standard deviation: $sigma\_z\_value\ = 25\, meters$;
- Sensors reveal the entity up to a certain maximum distance: $max\_detection\_range\ = 250\, meters$;
- Probability of not detecting the entity: $entity\_misdetection\_prob\ = 0.001$;
- Human model noise standard deviation: $sigma\_noise\_human\ = 10\, meters$;
- Animal model noise standard deviation: $sigma\_noise\_animal\ = 10\, meters$;
- Probability of the animal of changing direction: $animal\_change\_direction\_prob\ = 1/3$;
- Kalman filters iterations: $iterations\ = 30$;
- Minimum number of iterations for computing a delta between positions in order to compute direction and norm of the control input: $min\_iterations\_for\_delta\ = 5$;
- The scaling factor on the step that the animal will take in the previous direction: $animal\_model\_scaling\_factor\ = 3$.

V. Results

With our approach using a default set of hyperparameters described in the implementation section, we solved the classification problem with two DKFs implemented using the expressions derived as presented in the solution chapter and we obtained as results 90.4% accuracy on the human entities and 91.1% on the animal entities. The mean tracking error was $20.27\, meters$ on the human entities and $13.22\, meters$ on the animal entities. In this section, we show the results obtained by changing one parameter at a time from the default configuration defined in the previous section. With these variations, we are able to see how the hyperparameters affect the performance of the classification. The plots show the results of 1000 experiments for each tested configuration and class, since we wanted to have sufficiently statistically relevant results.

In *Fig. 1*, it can be seen how the algorithm seems not affected by the number of nodes in the sensor network. Indeed, the accuracy does not change in a significant way between the tests, but we should require a sufficient number of sensors of about 25 sensors in order to cover the whole area.

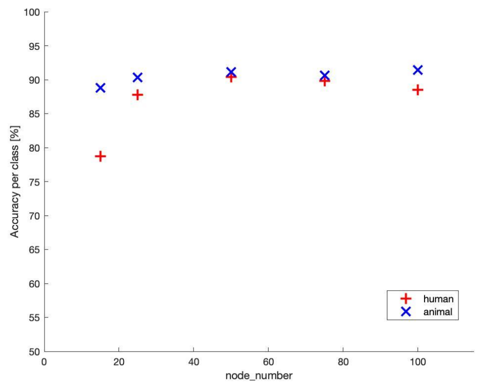

Fig. 1. How the True positives (human) and the True negatives (animal) are affected by the number of sensors in the network.

In *Fig. 2*, it can be seen how increasing the standard deviation of the sensor noise increases the overall accuracy of the classification. This may seem counterintuitive since it is expected that increasing the measurement noise standard deviation the quality of the results decreases. This happens because when the measurement noise standard deviation gets large the system model weights more, so the classes are more easily distinguishable. The confidence of the prediction gets affected too: the *human* and the *animal* classes improve almost linearly the confidence augmenting the measurement noise standard deviation.

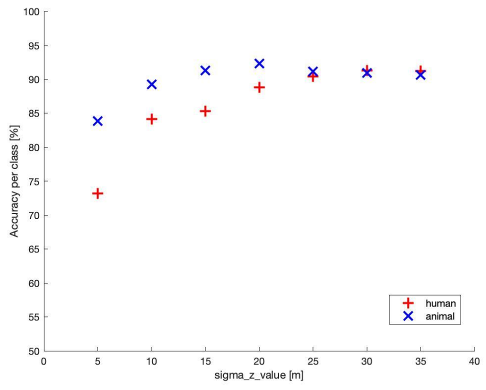

Fig. 2. True positive and true negative w.r.t. the variation of sigma_z_value through which the simulation is performed.

In *Fig. 3*, it can be seen that the accuracy of the classification task drops significantly when the model noise standard deviations increase. This happens because when the model noise standard deviations get large the classes are no more easily distinguishable. The confidence of the prediction gets affected too: the *human* and the *animal* classes decrease almost linearly the confidence augmenting the model noise standard deviations.

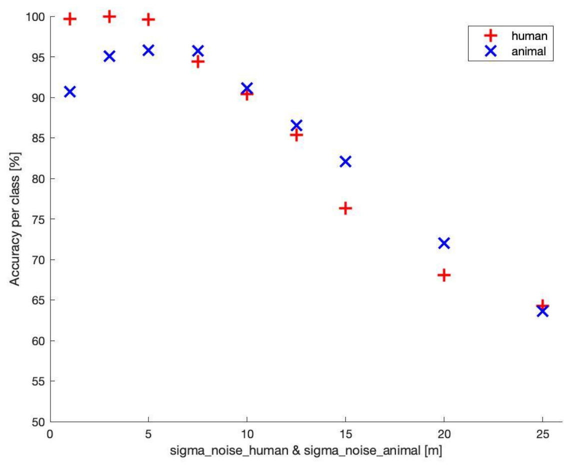

Fig. 3. How the True positives (human) and the True negatives (animal) are affected by model noise standard deviations.

In *Fig. 4,* it can be seen that the tracking error decreases when the number of nodes increases, this is because the measurements using the consensus algorithm are averaged on a larger number of samples reducing their error.

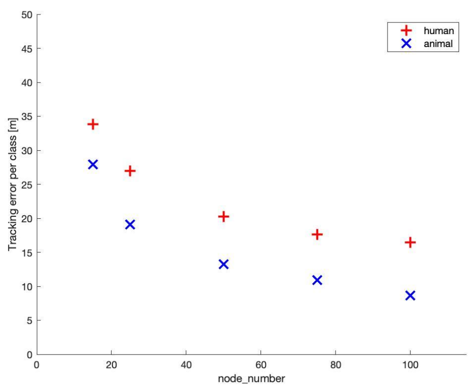

Fig. 4. Tracking error with respect to the number of nodes.

In *Fig. 5*, it is shown how the tracking error changes with respect to the measurement noise standard deviation. As expected the higher the measurement noise standard deviation the higher the tracking error.

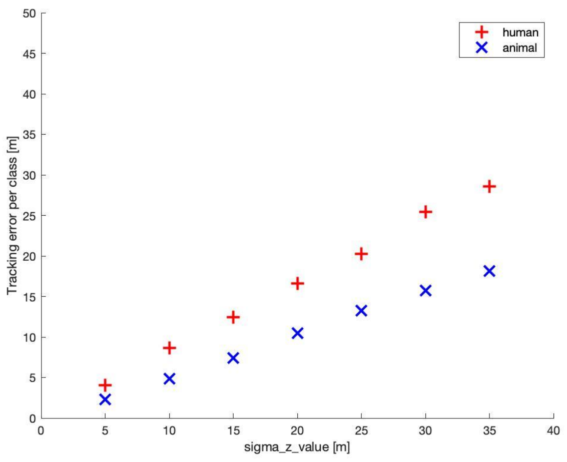

Fig. 5. Tracking error with respect to the variation of sigma_z_value.

In *Fig. 6*, it is shown that the tracking error decreases with respect to the human and animal noise standard deviations. This is due to the fact that when the model noise standard deviations increase the measurements are more important leading to a better precision in tracking the entity.

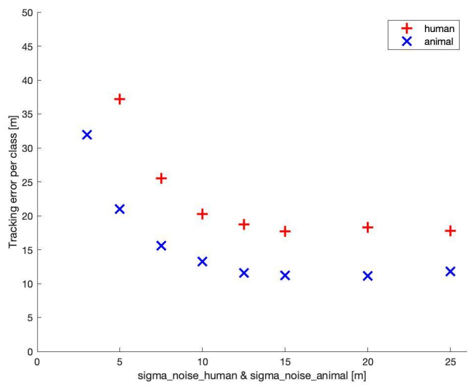

Fig. 6. Tracking error with respect to model noise standard deviations.

While testing different hyperparameter configurations we tried to create the best scenario with maximum communication and detection capabilities that can produce the best performances. We noticed that using low noise standard deviations for both measurements and the models help in boosting the performances. With the following configuration we obtained 98.65% accuracy over 1000 experiments for each class, 100% accuracy on the human entities and 97.3% on the animal entities. The mean tracking error was $2.54\,meters$ on the human entities and $3.40\,meters$ on the animal entities. The hyperparameters that were changed with respect to the default configuration are the following:

- $sigma\_z\_value\ = 15\,meters$;
- $sigma\_noise\_human\ = 5\,meters$;
- $sigma\_noise\_animal\ = 5\,meters$;
- $max\_communication\_range\ = 500000\ meters$;
- $communication\_failure\_prob\ = 0$;
- $max\_detection\_range\ = 500000\,meters$;
- $entity\_misdetection\_prob\ = 0$.

We also noticed that decreasing the measurement standard deviation up to reach the model ones to $1\,meter$, we obtained both an exceptional tracking and higher classification accuracy. With the following configuration we obtained 99.2% accuracy over 1000 experiments for each class, 100% accuracy on the human entities and 98.4% on the animal entities. The mean tracking error was $0.18\,meters$ on the human entities and $0.34\,meters$ on the animal entities. The hyperparameters that were changed with respect to the default configuration are the following:

- $sigma\_z\_value\ = 1\,meter$;
- $sigma\_noise\_human\ = 1\,meter$;
- $sigma\_noise\_animal\ = 1\,meter$;
- $max\_communication\_range\ = 500000\ meters$;
- $communication\_failure\_prob\ = 0$;
- $max\_detection\_range\ = 500000\,meters$;
- $entity\_misdetection\_prob\ = 0$.

VI. Conclusions

In this report, two Distributed Kalman Filters (DKFs) have been developed to track two different classes, *human* and *animal*, based on the knowledge of their model. These DKFs have been used to perform a classification task between the two models. The class associated with the highest probability of fitting the measurements over time with respect to the pdfs outputted by the two Kalman filters will be the one predicted by our distributed system. The resulting algorithm reaches an accuracy of 90% of predicting *true humans* and 91% of predicting *true animals* with the default configuration of hyperparameters.

We also tested different hyperparameter configurations and the method has proven to be resistant to changes in the hyperparameters, even when noise standard deviations are increased. We noticed a clear tradeoff between tracking the entity and classifying it when tuning the noise standard deviations for measurements and models. We tried to create the best scenario with maximum communication and detection capabilities, using lower noise standard deviations for both measurements and the models and we obtained 99.2% accuracy over 1000 experiments for each class, 100% accuracy on the human entities and 98.4% on the animal entities.

As we already said, distributed estimation allowed us to obtain better results thanks to the fact that many different sensors in different positions share their knowledge about the estimated position of the entity in order to reach a consensus with the others. In this way eventual noise collected can be attenuated if not canceled. The main benefit of having nodes that share knowledge with each other instead of having a centralized node that collects and elaborates information is an increased fault tolerance of the network.

This approach also has some downsides, first of all it is necessary to have a good knowledge of the dynamics of the entities. It is also important to have a good range of communication in order to share with more sensors the own measurements and reach a consensus faster. It is crucial then to have a good computer vision algorithm (e.g., a neural network properly trained on this task) able to output the estimated distance and direction of the entity with respect to the position of the sensor starting from the thermal camera data, with a good range of detection and with a high precision.

A possible future direction is to deploy our system on some sensors in a real scenario and test its effectiveness also with real entities moving in the environment under analysis. This approach is also limited by the fact that it can detect only one entity at the time, as future work it should be possible to track more entities and classify all of them simultaneously.

References

[1] Bar-Shalom, Yaakov, X. Rong Li, and Thiagalingam Kirubarajan. 2001. “The Multiple Model Approach.” In *Estimation with Applications to Tracking and Navigation: Theory Algorithms and Software*, 441-470. N.p.: John Wiley & Sons Inc.

[2] Li, Wenling, and Yingmin Jia. 2012. “Consensus-Based Distributed Multiple Model UKF for Jump Markov Nonlinear Systems.” *IEEE Transactions on Automatic Control*, January, 2012, 230-236.

[3] Olfati-Saber, R. 2007. “Distributed Kalman Filtering for Sensor Networks.” *Proceedings of the 46th IEEE Conference on Decision and Control*, December, 2007, 5492-5498.

The authors have the following addresses: {samuel.bortolin, alessandro.grassi}@studenti.unitn.it.

This report is the final document for the course of “Distributed Robot Perception”.