# Multiple Object Tracking Intern Report

Hasan Atakan Bedel
Department of Electrical and Electronics Engineering
Middle East Technical University

CONTENTS

## List of Figures



## List of Tables

# Multiple Object Tracking Intern Report

## I. Introduction

Multiple object tracking is an essential part of the perception pipeline. In order to solve this problem many methods have been developed. My internship subject was to approach this problem using classical methods which are proposed in the master thesis provided to me. To be more precise used methods are: IMM(Interaction Multiple Model) for modeling object trajectories with different kinematic models, UKF(Unscented Kalman Filter) for nonlinear filtering of the obtained measurements and JPDAF(Joint Probability Data Association Filter) for a sophisticated data association method. I have implemented all of them in python and prepared mock data to assess their results.

## II. Problem Definition and Approach

The objective of multiple object tracking is to link and filter object detections across the sampled timeline. This way each object's trajectory and state estimation can be obtained. The difference between single object tracking and multiple object tracking is, later needs to associate object detections with existing tracks so that their states can be estimated correctly based on these associted detections.

The supplied master thesis uses UKF(Unscented Kalman Filter) to estimate the object states. Kalman filter needs a prediction model to predict the next state. In the thesis multiple kinetic models are used; Constant Velocity(CV), Constant Turn-Rate Velocity(CTRV) Model and Random Motion Model(RM). CTRV has nonlinear nature, hence the need for UKF. In order to use these different models to give only one unified estimation IMM is used.

In order to track multiple objects an data association method is required. JPDAF is used for this purpose. JPDAF is a soft data association method, meaning it does not associate the detections with existing tracks one to one. It associates all detections to all tracks based on their calculated joint association probabilities.

## III. Objectives

To be honest, no clear objectives were provided in the beginning of my internship, only the master thesis. After have read the thesis and searched internet for some context I have determined the objectives myself, which were seem to be approved by my supervisor Dr. Berker Logoglu. Here is the list.

- Implementation of IMM, UKF, JPDAF algorithms
- Development of an test environment for the algorithms using mock data
- Test of the algorithms on Nuscenes dataset

## IV. Project Structure

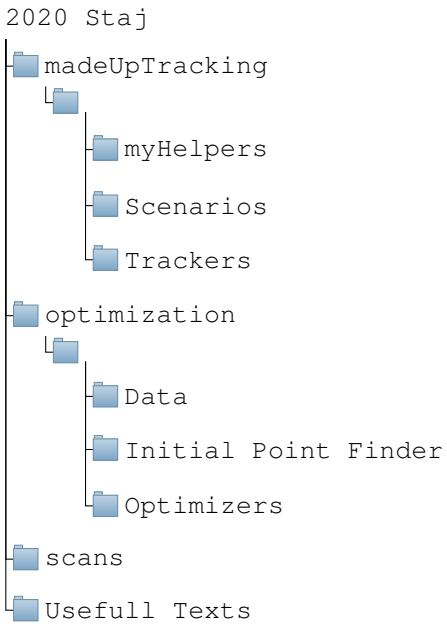


**Fig. 1:** *Project Structure*

## V. IMM-UKF-JPDAF

In this section, I will give details about the algorithms and their implementation details with some insights I have gained.

### A. IMM

IMM is used to integrate multiple kinematic models to obtain fused estimate. All used models have their own weights(mode probabilities), which determine their influence on the fused estimate. Mode probabilities are updated at each measurement time step, based on if one mode(kinematic model) were more successfull to estimate at the previous time step then the other modes.

| Notation | |
|---|---|
| $r_k$ | The mode at time step k |
| $z_k$ | Measured state at k |
| $x_k$ | State at k |
| $\hat{x}_k$ | State mean at k |
| $\hat{z}_k$ | State measured mean at k |
| $\hat{x}_k^i$ | State mean at k according to mode i |
| $\hat{x}_k^{[i]}$ | Mixed state mean at k for mode i |
| $S_k$ | State innovation matrix at k |
| $Z_k$ | The measurement at time k |
| $\mu_k^i$ | Mode probability at k |

**TABLE I:** *IMM Notation Table*

*1) Mixing:* At each step all the models have to mix their states with each other. This is the interaction step where IMM(Interaction Multiple Model)'s 'I' take it's name. This step helps models to stay in some bound(not formal) without diverging from each other completely.

$$\mu_{k-1|k-1}^{ji} \triangleq P(r_{k-1} = j \,|\, r_k = i, z_{0:k-1}) \tag{1}$$

$$\mu_{k-1|k-1}^{ji} = \frac{\pi_{ji}\,\mu_{k-1}^{j}}{\sum\limits_{l=1}^{N_r} \pi_{li}\,\mu_{k-1}^{l}} \tag{2}$$

$$\hat{x}_{k-1|k-1}^{[i]} = \sum_{j=1}^{N_r} \mu_{k-1|k-1}^{ji}\,\hat{x}_{k-1|k-1}^{j} \tag{3}$$

$$\begin{aligned} P_{k-1|k-1}^{[i]} = \sum_{j=1}^{N_r} \mu_{k-1|k-1}^{ji}\,[P_{k-1|k-1}^{j} + \\ (\hat{x}_{k-1|k-1}^{j} - \hat{x}_{k-1|k-1}^{[i]})\,(\hat{x}_{k-1|k-1}^{j} - \hat{x}_{k-1|k-1}^{[i]})^T] \end{aligned} \tag{4}$$

*2) Mode Prediction Updates:* In this part each model independently predicts the next state, which are state mean and state covariance.

*3) Mode Measurement Updates:* Again in this part each model independently updates their state using the measurement(s) given.

*4) New Mode Probabilities:* Here each model's mode probability is calculated. New mode probabilities are dependent on how good each model were able estimate the new state, markov chains and previous mode probabilities.

$$\mu_k^i = \frac{N(Z_k; \hat{z}_{k|k-1}^i, S_k^i) \sum\limits_{j=i}^{N_r} \pi_{ji}\,\mu_{k-1}^{j}}{\sum\limits_{l=1}^{N_r} N(Z_k; \hat{z}_{k|k-1}^l, S_k^l) \sum\limits_{j=1}^{N_r} \pi_{jl}\,\mu_{k-1}^{j}} \tag{5}$$

*5) Output Estimate Calculation:* Finally the one unified estimate is calculated based on the new calculated mode probabilities. Each model can contribute to the unified estimate proportional to their mode probability. Note that this stage does not affect any part of the algorithm, it is only used for output.

$$\hat{x}_{k|k} = \sum_{i=1}^{N_r} \mu_k^i\,\hat{x}_{k|k}^i \tag{6}$$

$$P_{k|k} = \sum_{i=1}^{N_r} \mu_k^i\,[P_{k|k}^i + (\hat{x}_{k|k}^i - \hat{x}_{k|k})\,(\hat{x}_{k|k}^i - \hat{x}_{k|k})^T] \tag{7}$$

### *B. UKF*

It is used to estimate state for non-linear prediction models. Best intuitive and formal explaination is in it's original paper [3].

| Notation | |
|---|---|
| $\hat{x}_{k-1\|k-1}^i$ | i'th sigma point obtained from k-1'th step |

**TABLE II:** *UKF Notation Table*

*1) Unscented Transform:* First sigma points are calculated. It is like sampling near the state mean, but in a systematic and proven way. Note that $(\sqrt{(L+\lambda)\,P_{k-1|k-1}})_i$ is the i'th column or row of the matrix inside the parentheses [5]. But one must stick to one choice, row or column.

$$\begin{aligned} \hat{x}_{k-1|k-1}^0 &= \hat{x}_{k-1|k-1} \\ \hat{x}_{k-1|k-1}^i &= \hat{x}_{k-1|k-1} + (\sqrt{(L+\lambda)\,P_{k-1|k-1}})_i \\ &\qquad i \in 1, ..., L \\ \hat{x}_{k-1|k-1}^i &= \hat{x}_{k-1|k-1} - (\sqrt{(L+\lambda)\,P_{k-1|k-1}})_{i-L} \\ &\qquad i \in L+1, ..., 2L \end{aligned} \tag{8}$$

*2) Prediction:* At this part each sigma points are passed through the prediction function. After that, each points are used to get the predicted state mean and covariance. The used parameter values in my implementation for the parameters presented here($\alpha, \kappa, \beta ...$) can be found in the original paper [3].

$$
\begin{aligned}
&\hat{x}^i_{k|k-1} = f(\hat{x}^i_{k-1|k-1}),\ \ i \in 0, ..., 2L \\
&W^0_s = \frac{\lambda}{L+\lambda},\ W^0_c = \frac{\lambda}{L+\lambda} + (1 - \alpha^2 + \beta) \\
&W^i_s = W^i_c = \frac{1}{2(L+\lambda)},\ \lambda = \alpha^2(L+\kappa) - L \\
&\hat{x}_{k|k-1} = \sum_{i=0}^{2L} W^i_s\ \hat{x}^i_{k|k-1} \\
&P_{k|k-1} = \sum_{i=0}^{2L} W^i_c\ [\hat{x}^i_{k|k-1} - \hat{x}_{k|k-1}]\ [\hat{x}^i_{k|k-1} - \hat{x}_{k|k-1}]^T + Q
\end{aligned}
\tag{9}
$$

*3) Second Unscented Transform:* In prediction state the state covariance matrix is calculated and now it is time to find the new sigma points with respect to this new predicted state mean and covariance. Why do we need these sigma points? Reason is to make measurements on these sigma points. If the measurement function is linear than this part and next part is not necessary. Update parameters($S_k, K_k$) can be calculated using the classical kalman filter equations. This is the case with my project.

$$
\begin{aligned}
&\hat{x}^{*0}_{k|k-1} = \hat{x}_{k|k-1} \\
&\hat{x}^{*i}_{k|k-1} = \hat{x}_{k|k-1} + \quad (\sqrt{(L+\lambda)\ P_{k|k-1}})_i \\
&\qquad\qquad\qquad\qquad i \in 1, ..., L \\
&\hat{x}^{*i}_{k|k-1} = \hat{x}_{k|k-1} - \quad (\sqrt{(L+\lambda)\ P_{k|k-1}})_{\,i-L} \\
&\qquad\qquad\qquad\qquad i \in L+1, ..., 2L
\end{aligned}
\tag{10}
$$

*4) Calculate Update Parameters:* If the measurement function is linear, which is the case for my situation, than this part can be replaced with classical kalman filter update parameter calculations. In my implementation I have coded both of them to see the difference. Note that I have not seen any formal explaination to do this but it made sense and indeed they give the exact same result.

$$
\begin{aligned}
&\hat{z}^i_k = h(\hat{x}^i_{k|k-1}),\ i \in 0, ..., 2L \\
&\hat{z}_{k|k-1} = \sum_{i=0}^{2L} W^i_s\ \hat{z}^i_k \\
&P_{z_{k|k-1}, z_{k|k-1}} = \sum_{i=0}^{2L} W^i_c\ [\hat{z}^i_{k|k-1} - \hat{z}_{k|k-1}]\ [\hat{z}^i_{k|k-1} - \hat{z}_{k|k-1}]^T \\
&S_k = P_{z_{k|k-1}, z_{k|k-1}} + R \\
&P_{x_{k|k-1}, z_{k|k-1}} = \sum_{i=0}^{2L} W^i_c\ [\hat{x}^i_{k|k-1} - \hat{x}_{k|k-1}]\ [\hat{z}^i_{k|k-1} - \hat{z}_{k|k-1}]^T \\
&K_k = P_{x_{k|k-1}, z_{k|k-1}}\ S^{-1}_k
\end{aligned}
\tag{11}
$$

*5) Update:* These are usual kalman filter update equations.

$$
\begin{aligned}
&\hat{x}_k = \hat{x}_{k|k-1} + K_k\ (Z_k - \hat{z}_{k|k-1}) \\
&P_k = P_{k|k-1} - K_k\ S\ K^T_k
\end{aligned}
\tag{12}
$$

### C. JPDAF

JPDAF is used for soft data association. It assumes that the number of existing tracks is a known, meaning it does not have power to initiate new tracks from the given measurements. It calculates the possibility of associations between each track and each measurement. This data association probability is the output of the JPDAF. Later these probabilities are passed to PDAF along with the measurements to update the each tracker. JPDAF is clearly explained in [1], but implementation part was not clear.

*1) Validation Matrix:* "Validation Matrix" is presented here [1, p.312]. It is a binary matrix with $m_k$ number of rows and $N_r+1$ number of columns. The elements of the matrix, lets say $b_{i,j}$, represent if the i'th measurement is close to j'th track. If it is close, that element is 1 otherwise 0. Whether a measurement is close to a track is decided based on the mahalonobis distance between the track's measured state mean and the measurement, using the innovation matrix as the covariance. The rationale can be find in [1, p.312]. Note that $j = 0$ represents the possibility of that measurement is associated with no track. To include this possibility for all measurements the first column of the validation matrix is initiated as 1.

*2) Association Events:* An association event is nothing but just one of the possibilites to associate the existing tracks and the incoming measurements. In my implementation I have written an exhaustive algorithm($O(m_k^{N_r+1})$) to try find the possible association events using the validation matrix. If the validation matrix is dense, meaning the all measurements are close to all tracks, and if there are vast number of tracks and measurements, the performance of the algorithms is very bad. I measured the run time of my implementation with $m_k = 8$ and $N_r = 8$ and filling the validation matrix with all ones, as $41.797$ seconds, which is very bad. On the other hand with $m_k = 5$ and $N_r = 5$ and filling the validation matrix with

all ones, as $0.036$ seconds. Note that a more sparse validation matrix really helps to time performance, but downside is the sparsity can not be guaranteed.

*3) Joint Association Probabilities:* All possible events were generated and stored. Now it is time to calculate their probability. Now here I give the formal definition of an event $\theta$.

$$\theta = \bigcap_{j=1}^{m} \theta_{j,t_j} \tag{13}$$

where $\theta_{j,t_j}$ is the event that measurement j is associated with track $t_j$. Track $t_j$ is the associated track with measurement j. Now here are the two ways of calculating the probability of an event.

Parametric calculation:

$$P(\theta|Z^k) = \frac{1}{c} \prod_{j=1}^{m_k} \{\lambda^{-1} f_{t_j}[z_j(k)]\}^{\tau_j} \prod_{t=1}^{N_r} (P_D)^{\delta_t} (1-P_D)^{1-\delta_t} \tag{14}$$

Nonparametric calculation:

$$P(\theta|Z^k) = \frac{1}{c'} \phi! \prod_{j=1}^{m_k} \{V f_{t_j}[z_j(k)]\}^{\tau_j} \prod_{t=1}^{N_r} (P_D)^{\delta_t} (1-P_D)^{1-\delta_t} \tag{15}$$

$$f_{t_j}[z_j(k)] = N[Z_k^j;\ \hat{z}_{k|k-1}^{t_j},\ S_k^{t_j}] \tag{16}$$

The derivations can be found in [1, p.317].

| Notation | |
|---|---|
| $V$ | Volume of the surveillance region |
| $\lambda$ | Spatial density of false measurements |
| $P_D$ | Detection probability of the detector |
| $\delta_t$ | A binary number showing whether target t is associated with a measurement in that event |
| $\tau_j$ | A binary number showing whether measurement j is associated with a target in that event |
| $t_j$ | It is the index of the track which measurement j is associated with in that event |
| $\phi$ | The number of false alarms in that event |

**TABLE III:** *JPDAF Notation Table*

The implementation of surveillance region was not clear in [1]. One solution could be to take the volume of the track with highest volume [1, p.211], or maybe summing the max volumes of all tracks could be a solution. That is why I haven't implemented the non-parametric version, it is ambiguous.

Note that the calculation of event probabilities are not normalized(there is a $\frac{1}{c}$ constant in the event probability calculation). That is why we need to normalize the event probabilities now.

$$P(\theta|Z^k) = \frac{P(\theta|Z^k)}{\sum\limits_{\theta' \in all\ events} P(\theta'|Z^k)} \tag{17}$$

After normalization it is time to calculate the data association probabilities needed for the PDAF part.

$$\beta_k^{jt} \triangleq P(\theta_{jt}|Z_k) = \sum_{\theta:\ \theta_{jt} \in \theta} P(\theta|Z_k) \tag{18}$$

where $\theta_{jt}$ is the event that measurement j is associated with target t.

Equation(18) is for $j \in 1, ..., m_k$. For j=0, meaning no measurement is associated with the track t, here is the calculation.

$$\beta_k^{0t} = 1 - \sum_{j=1}^{m_k} \beta_k^{jt} \tag{19}$$

### D. PDAF

PDAF is about updating a track t with multiple measurements each of which has a association probability with that track t. In fact, one of the measurements is "no measurement available for track" which also has a probability [1, p.130].

Note that, I will omit the superscript "t" from the equations, which would represent the equations are for track t.

Here are some variables to be used in the state update equations.

$$\begin{aligned} v_k^j &= Z_k^j - \hat{z}_{k|k-1} \\ v_k &= \sum_{j=1}^{m_k} \beta_k^{jt}\, v_k^j \\ P_{k|k}^c &= P_{k|k-1} - K_k\, S_k\, K_k^T \\ \tilde{P}_k &\triangleq K_k\, [\sum_{j=1}^{m_k} \beta_k^{jt}\, v_k^j\, (v_k^j)^T \;-\; v_k\, v_k^T]\; K_k^T \end{aligned} \tag{20}$$

Now the state update equations.

$$\begin{aligned} \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k\, v_k \\ P_{k|k} &= \beta_k^{0t}\, P_{k|k-1} + [1 - \beta_k^{0t}]\, P_{k|k}^c + \tilde{P}_k \end{aligned} \tag{21}$$

The explanation and derivation can be find in [1, p.130].

### E. Putting It All Together

I will now summarize how to use all these algorithms together. Note that only change will be in IMM algorithm in the calculation of new mode probabilities.

*1) Mixings:* This part is not different. For each track, independently from other tracks, different models will mix their states based on their mode probabilities.

*2) Predictions:* This part is not different. Each track's, independently from other tracks and independent from other models inside that track, will predict the next state using UKF algorithm.

*3) Joint Data Association Probabilities Calculation:* This part is tricky with IMM. Each track should have a state, whereas with IMM they have models. That is why for the event probability calculations one needs to fuse these models to get one state for each track. This fusing is same as the output stage of the IMM, but using the predictions of the models and the previous stage mode probabilities, at least this is what I have implemented. I could not find any information how to implement this part elsewhere.

*4) Updates:* In this part using the data association probabilities and measurements calculated from JPDAF stage, each track's models will update their states.

*5) New Mode Probabilities Calculation:* This is the part where new equation is needed, since there isn't just one measurement but measurements, hence mode probability calculation must be altered. Here is the equations from [1, p.211] and [4, p.53].

$$\mu_k^i = \frac{\wedge_k^i[Z_k] \sum_{j=1}^{N_r} \pi_{ji}\, \mu_{k-1}^j}{\sum_{l=1}^{N_r} \{\wedge_k^l[Z_k] \sum_{j=1}^{N_r} \pi_{jl}\, \mu_{k-1}^j\}} \tag{22}$$

where

$$\wedge^j[Z_k] = \frac{1 - P_D}{V^{m_k}} + \frac{P_D}{m_k\, V^{m_k - 1}} \sum_{i=1}^{m_k} N[Z_k^i;\, \hat{z}_{k|k-1}^j,\, S_k^j] \tag{23}$$

Note that, in my implementation I have used the highest volume among the models as the volume V, surveillance region. The volume calculation can be found here [1, p.130].

*6) Final Output Estimations:* Each track will fuse their models using the new calculated model probabilities to get an updated estimate for their states.

### F. Testing of the Algorithms

In order to verify and validate my implementation I have created mock data and implemented tracker classes to track these generated data.

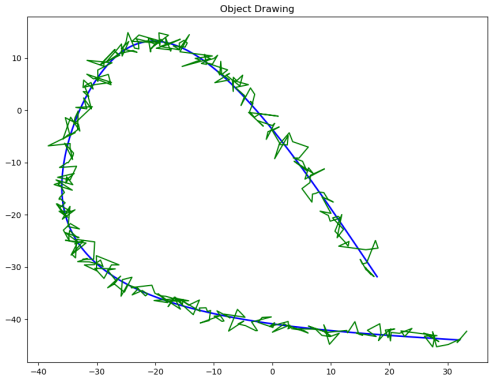


**Fig. 2:** *This is an image of a randomly generated scene. Blue path is ground truth and green lines are noisy measurements generated by adding gaussian noise added upon to the ground truth.*

Unfortunately, I did not have time to test them on a real data set.

## VI. Trackers

I have implemented 4 different tracker classes to test my algorithms. I have run them on randomly generated scenes to see if they actually work or how good they work.

Note that I have implemented track initialization based on the simple algorithm presented here, [2, p.247]

### A. Single Object Single Model Tracker

This class can be found inside "madeUpTrackers/Trackers/SingleTarget/allMe" folder.

It is used to track single object with single model. The measurement to be fed to the tracker in each sampling time is expected to exist and be one. Usage of this class can be found in "demo_singleTarget.py".

The algorithm of the interest to be validated is *UKF*. I have implemented 3 different models; Constant Velocity(CV), Constant Velocity Turn Rate(CVTR), Random Motion(RM). I have implemented CV in two different ways, both with classic kalman filter and unscented kalman filter, since CV is a linear model. I have checked different implementations online(e.g. filterpy) to make sure I have implemented correctly. Both comparing with different implementation and testing it on mock data, I have concluded, it is correct.

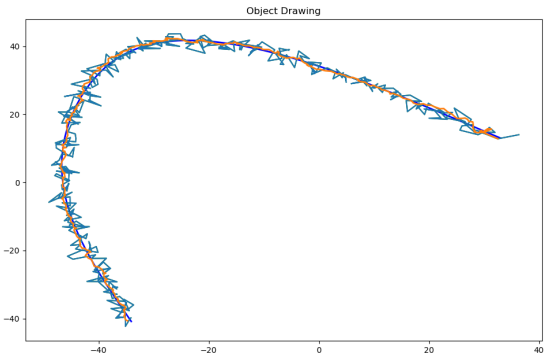


**Fig. 3:** *This is the result of the tracking using "Single Object Single Model" tracker class with model "CV" using unscented kalman implementation on a randomly generated scene. Light blue line is measurements dark blue line is ground truth and orange line is the estimates of the tracker*

### B. Single Object Multiple Model Tracker

This class can be found inside "madeUpTrackers/Trackers/SingleTarget/allMe" folder.

It is used to track single object with multiple model. The measurement to be fed to the tracker in each sampling time is expected to exist and be one. Usage of this class can be found in "demo_singleTarget.py".

The algorithms of the interest to be validated are *UKF* and *IMM*. I have combined "CV" and "CVTR" models. Previously I have added "RM" model too, but the results were not satisfying. In the master thesis it was added, maybe after process noise optimization it would give nice results. You can change the class to add more models. Just add new model forward and measurement functions. Also you would need to change the markov chain matrix and initial mode probabilities. Make sure that sum of the initial mode probabilities add up to 1.

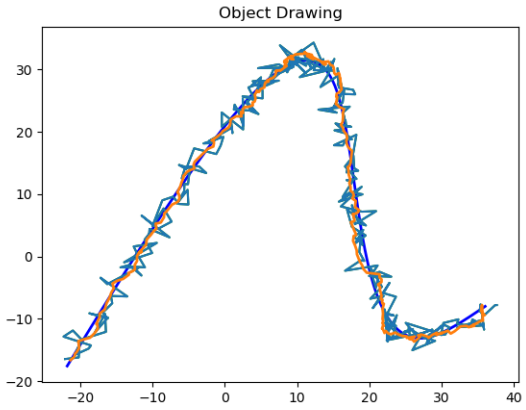


**Fig. 4:** *This is the result of the tracking using "Single Object Multiple Model" tracker class with models "CV" and "CVTR" combined using unscented kalman implementation on a randomly generated scene. Light blue line is measurements dark blue line is ground truth and orange line is the estimates of the tracker*

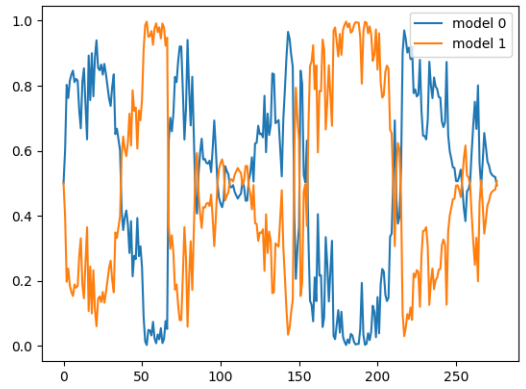


**Fig. 5:** *Here are the model probabilities of the imm tracker in figure 4. The legend "model 0" corresponds to "CV" model, and "model 1" corresponds to "CVTR" model.*

### C. Multiple Object Single Model Tracker

This class can be found inside "madeUpTrackers/Trackers/MultipleTarget/allMe" folder.

It is used to track multiple object with single model. The measurements to be fed to the tracker in each sampling time can be of any number, including 0. Usage of this class can be found in "demo_multipleTarget_singleTarget.py".

The algorithms of the interest to be validated are *UKF* and *JPDAF*. I have implemented 3 different models; Constant Velocity(CV), Constant Velocity Turn Rate(CVTR), Random Motion(RM). I have implemented CV in two different ways, both with classic kalman filter and unscented kalman filter, since CV is a linear model. I could not find any reliable source for the JPDAF implementation. On the other hand, results on mock data indicate my JPDAF implementation is correct.

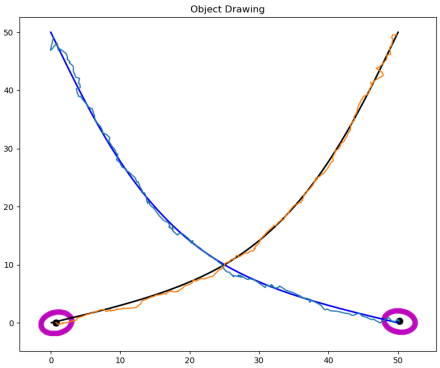


**Fig. 6:** *This is the result of the tracking using "Multiple Object Single Model" tracker class with model "CVTR" using unscented kalman implementation on a randomly generated scene. In this plot the measurements are not shown. Purple circles shows the gating region of the trackers, the regions where trackers are allowed to associate measurements with.*

### *D. Multiple Object Multiple Model Tracker*

This class can be found inside "madeUpTrackers/Trackers/MultipleTarget/allMe" folder.

It is used to track multiple object with multiple model. The measurements to be fed to the tracker in each sampling time can be of any number, including 0. Usage of this class can be found in "demo_multipleTarget_multipleTarget.py".

The algorithms of the interest to be validated are *UKF*, *IMM* and *JPDAF*. I have implemented 3 different models; Constant Velocity(CV), Constant Velocity Turn Rate(CVTR), Random Motion(RM). Although the results of imm with single target were satisfactory, this is not the case for multiple target.

Sometimes, the predictions of the tracker increases the state covariance too much, causing the tracker to diverge. Another issue is that, there are high frequency oscillations in the tracker paths. I did not have enough time to investigate these problems. I suspect, these problems would be resolved if process noise and some other parameters in the algorithms were set right. But it could also be an implementation error, which I have checked several times and could not find.

## VII. The Need For Optimization

Each model needs to have a process noise matrix, which is added to the state prediction covariance matrix in every step,

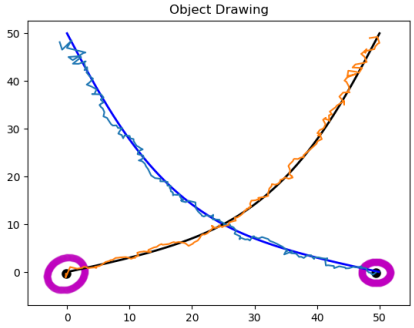


**Fig. 7:** *This is the result of the tracking using "Multiple Object Multiple Model" tracker class with models "CV" and "CVTR" using unscented kalman implementation on a randomly generated scene. In this plot the measurements are not shown. Purple circles shows the gating region of the trackers, the regions where trackers are allowed to associate measurements with.*

accounts for the model's imperfectness. If the objects to be tracked behaved exactly like the models suggest, then we would not need to add process noise, but they don't and we have to add it.

In each iteration existing tracks' magnitude of the state covariances decreases. After few iterations their values become small enough to discard the information coming from the measurements. But if the process noise is added to these state covariances after each prediction, their magnitude does not converge to zero but to that of process noise. This way trackers don't become overconfident to discard the incoming measurements.

Obviously, the selected values in the process noise matrix are critical. Hand picking these values can be difficult, though it is not impossible. The dynamics of the interactions between these values can give rise to counter-intuitive results that makes it hard to do. This is why one needs an optimization technique to find the best process noise possible for each models for a given data-set or environment dynamics.

*1) Optimization:* I could not spend much time on finding the best optimization technique. Because of it's simplicity I have used the optimization technique used in [reference here] with a minor modification.

First, I find the process noise for each model using the method given in [reference]. Then, by multiplying this matrix with scalar values selected from a interval and plotting it, I decide for the best scale for each model. At first I have tried to use back-propagation to find this best scale, but it takes to much time, and does not give me chance to see a visualization of the loss space of the complete interval. Another idea, that I have came out with is to use this scaled process noise matrix

as a initial point for an back-propagation training of all the entries of the process noise. Again it took too much time to back-propagate, and I stopped working on it. If you want to work on this, make sure that with each iteration of the back-propagation, process noise matrix stays positive-definite. The way to do is generating(Cholesky decomposition) process noise from a lower or a upper triangular matrix with positive diagonal entries. While back-propagating to the lower(or upper) matrix, make sure that diagonal entries stay positive, clamp the elements of the gradient if it makes it negative. I had used the L2 distance between the updated estimates of the track and ground-truth as my loss function.

Here is how I calculated the initial process noise covariance matrix for model "CVTR", based on the technique in [reference].

First write down the model's prediction equations.

$$
\begin{aligned}
x_{k+1|k} &= x_k + \frac{v_k}{\dot{\phi}_k}\ (-sin(\phi_k)\ +\ sin(\Delta t * \dot{\phi}_k + \phi_k)) + \boldsymbol{q_x} \\
y_{k+1|k} &= y_k + \frac{v_k}{\dot{\phi}_k}\ (cos(\phi_k)\ -\ cos(\Delta t * \dot{\phi}_k + \phi_k)) + \boldsymbol{q_y} \\
\phi_{k+1|k} &= \phi_k + \Delta t * \dot{\phi}_k + \boldsymbol{q_\phi} \\
v_{k+1|k} &= v_k + \boldsymbol{q_v} \\
\dot{\phi}_{k+1|k} &= \dot{\phi}_k + \boldsymbol{q_{\dot{\phi}}}
\end{aligned}
\tag{24}
$$

Now leave the process noises alone and find the variance of the right hand side by approximating the states using ground-truth data. Here are the approximations of the states. I will use capital letter to denote the ground truth data.

$$
\begin{aligned}
x_k &\approx X_k \\
y_k &\approx Y_k \\
v_k &\approx \frac{\sqrt{(X_k - X_{k-1})^2 + (Y_k - Y_{k-1})^2}}{\Delta t} \\
\phi_k &\approx tan^{-1}(\frac{Y_k - Y_{k-1}}{X_k - X_{k-1}}) \\
\dot{\phi}_k &= \frac{\phi_k - \phi_{k-1}}{\Delta t} \\
x_{k+1|k} &\approx X_{k+1} \\
y_{k+1|k} &\approx Y_{k+1} \\
v_{k+1|k} &\approx V_{k+1} \\
\phi_{k+1|k} &\approx \phi_{k+1} \\
\dot{\phi}_{k+1|k} &\approx \dot{\phi}_{k+1}
\end{aligned}
\tag{25}
$$

## VIII. Performance Assesment

| Time Performances | | | |
|---|---|---|---|
| Object Count | CV(Non-linear) Model | CV(Linear) Model | IMM(CV & CVTR) |
| 3 | 209 FPS | 441 FPS | 99 FPS |
| 4 | 152 FPS | 222 FPS | 59 FPS |
| 5 | 75 FPS | 104 FPS | 42 FPS |
| 7 | 22 FPS | 19 FPS | 15 FPS |

**TABLE IV:** *Tracker Performances Table*

When compared, linear model is faster than non-linear model and imm. But this gap shrinks when the number of objects increases, since the over-head of JPDAF algorithms start to dominate. When the number of objects increases, performance of the trackers decreases dramatically. Main reason is that, as the objects increases, number of joint association event count increases, hence the time spent on calculation of those events. Here I give the average percentage time spent on different parts of IMM-UKF-JPDAF tracker.

| Spended Time on Different Parts | | |
|---|---|---|
| Part | Average Time | Percentage |
| Prediction | 0.013s | 8.63% |
| New Track Check | 0.00113s | 0.74% |
| Construction of Validation Matrix | 0.000175s | 0.11% |
| Event Generation | 0.00732s | 4.83% |
| Event Probability Calculations | 0.112s | 74.50% |
| Update | 0.0169s | 11.16% |

**TABLE V:** *Spended Time for tracking 8 objects with 6fps*

## IX. Fullfilment of the Objectives

I have fulfilled my objectives partially. I have completed the implementation of the algorithms and the trackers using them step by step to test my implementations on mock data. But I could not test my trackers on a real data-set.

## X. Conclusion

I have completed the implementation and the integration of IMM, UKF and JPDAF algorithms and successfully used them on my tracker classes. I have tried to test my implementations on mock data. I was satisfied with the results of tracker classes; "singleTarget-singleModel", "singleTarget-multipleModel", "multipleTarget-singleModel". But I am not sure if "multipleTarget-multipleModel" class is working right. The problem might be the wrong integration of IMM with JPDAF, though I have spent time to make sure it is right. Other

possibility is parameters I have used are not tuned right. For this I did not spend the necessary time.

## REFERENCES


[1] Y. Bar-Shalom and X.-R. Li, *Multitarget-Multisensor Tracking Princibles and Techniques*, 3rd ed. 1995.

[2] Y. Bar-Shalom, X.-R. Li and T.Kirubarajan, *Estimation with Application to Tracking and Navigation*, New York, USA: John Wiley & Sons, Inc., 2001.

[3] E. a. Wan, R. van der Merwe, and A. Nelson, "Dual estimation and the unscented transformation," Nips, no. January 2000, pp. 1–7, 2000.

[4] A.S. Abdul Rachman, 2017. 3D-LIDAR Multi Object Tracking for Autonomous Driving, Multi-target Detection and Tracking under Urban Road Uncertainties. Delft University of Technology, Netherlands.

[5] Gabriel A. Terejanu. Unscented Kalman Filter Tutorial. University at Buffalo, Buffalo.