ANKARA SCIENCE UNIVERSITY, RESEARCHER
VOL. 1, NO. 1, JULY 2021

Implementation of the Multi Pedestrian Tracking Problem on Graphics Processing Unit

Özcan DÜLGER [1,2*]
[1] Department of Computer Engineering, Artvin Coruh University; ORCID: 0000-0001-7525-1064
[2] Department of Computer Engineering, Middle East Technical University
* Corresponding author: odulger@ceng.metu.edu.tr

Received: 30 June 2021; Accepted: 2 July 2021

Abstract

Particle filter is a serial Monte Carlo estimation method which is used when the system or the measurement model of the application is highly non-linear and uncertainties are large. As the number of particles increases, the computation cost of the particle filter increases. Graphics processing unit (GPU) offers promising solutions to accelerate the particle filter. Since there are many pedestrians in a multi pedestrian tracking problem, more than one particle filters run at a time. So, it is important to implement the particle filter on the GPU efficiently. In this study, we implement a multi pedestrian tracking algorithm on the GPU. We have three pedestrians along with some clutters (ten clutters at each time step). There may be up to 13 measurements which stand for too many particle filters run at a time. We use gating, association techniques in order to assign a measurement to a track (a pedestrian). We implement the particle filters on the GPU and achieve up to 9.79x speed up. When we consider the duration between two consecutive measurements is small, implementing the particle filter on the GPU becomes substantial as the number of particles increases. Furthermore, the quality of the particle filters is significant.

Keywords: multi pedestrian tracking, particle filter, Systematic resampling, graphics processing unit, CUDA programming, Tesla K40 board

1. Introduction

Particle filter is a serial Monte-Carlo method that is used efficiently in the applications whose system or measurement model is highly non-linear and uncertainties are large. The target that is tracked is represented by the particles and their weights. The estimation of the target is the weighted average of the particles. The computational cost of the particle filter increases as the number of particles increases which causes the real time implementation of the particle filter challenging. Graphics processing units offer promise for the real time implementation of the particle filter since they have many cores in their architectures [1].

As time progresses, the normalized weight of one particle approaches to one, the normalized weights of the remaining particles become nearly zero. In this situation, the remaining particles cause wasted computational effort. Further, the degeneracy problem arises because only one particle is effective. One of the solutions that are used to overcome degeneracy problem is resampling. In resampling, the weights of the particles are determiner in the results. The particles whose weights are large are replicated by the ratio of their weights and the particles whose weights are very small are eliminated. We use Systematic resampling algorithm in our experiments. It is very common resampling algorithm in literature [1][2].

Understanding the actions of the people in the public area is important for estimating their intentions and attentions or extracting knowledge about their behaviors and social networks. The body orientations or position information are used in these processes. With these processes, we can understand which exhibits they like in a museum or we can extract who are just walking and who are interested in shopping in a shopping mall [3]. In this study, we implement a multi pedestrian tracking algorithm by using particle filter as the estimation method. At first, we implement the multi pedestrian tracking algorithm serially with C++ programming language. Then we implement it with CUDA programming language on the graphics processing unit (GPU). We show the quality and the execution time performances of the implementations along with the speed up of the GPU implementation.

In the second section, we mention some of the studies about Systematic resampling and pedestrian tracking in literature. In section 3, we describe the sampling importance resampling (SIR) particle filter and the Systematic resampling methods. Moreover, we describe our multi pedestrian tracking algorithm in detail. In section 4, we give information about the experimental environment and discuss the experiment results. Section 5 concludes the study.

2. Literature

There are some studies in literature about the efficient implementation of Systematic resampling on the GPU [4-6]. There are also other studies in literature which track the pedestrians by using particle filter. Brscic and co-workers track the positions and the body directions of the pedestrians in a shopping mall. They use multiple 3D range sensors to get the observations of the pedestrians. They observe head position and the body angle as the measurement data of the person and use video sequences as a sensor input [3]. To weight the particles, they use hypothetical target representation. Furthermore, they compare the performances of the particle filter with CAMSHIFT and Mean-Shift algorithms [7]. Guan and co-workers propose an improved particle filter to track the pedestrians in surveillance video. They use color histogram model as the observation model [8]. There are also other studies that improve the multi pedestrian tracking algorithms [9][10]. Further, there are some studies that implement the multi pedestrian tracking algorithms on the GPU [11][12].

3. SIR Particle Filter and Multi-Pedestrian Tracking Algorithm

The stages of the sampling importance resampling (SIR) particle filter are given in Algorithm 1 [1].

In this algorithm, N is the number of particles; i is the index of the particle; k is the index of the time step; x_k^i represents the state of i-th particle; $\widetilde{w}_k^i$ is the weight of i-th particle; $sum\widetilde{w}$ is the sum of all weights; w_k^i is the normalized weights of i-th particle; x_k is the state of the target; $\hat{x}_k$ is the estimation of the target; z_k is the measurement; $p(x_k|x_{k-1}^i)$ is the transitional prior and $p(z_k|x_k^i)$ is the likelihood function.

Algorithm 1: SIR Particle Filter [1]

```
    [ ${x_k^i}_{i=1}^N$ ] = SIR [${x_{k-1}^i}_{i=1}^N$ , $z_k$]
1       foreach $i = 1:N$
                $x_k^i \sim p(x_k|x_{k-1}^i)$
                $\widetilde{w}_k^i = p(z_k|x_k^i)$
        end foreach
2       $sum\widetilde{w} =$ SUM [ ${\widetilde{w}_k^i}_{i=1}^N$ ]
3       foreach $i = 1:N$
                $w_k^i = sum\widetilde{w}^{-1}\widetilde{w}_k^i$
        end foreach
        $\hat{x}_k =$ SUM [ ${x_k^i * w_k^i}_{i=1}^N$ ]
4       [ ${x_k^i}_{i=1}^N$ ] = RESAMPLE [${x_k^i, w_k^i}_{i=1}^N$ ]
```

In the first stage, the state of each particle at time k is predicted by using transitional prior. And the weights of the particles are calculated by using measurement at time k and likelihood function. In the second stage, the sum of the weights is calculated. In third stage, the weights are normalized and the estimation of the target at time k is calculated. In the fourth stage, resampling is performed. There are dependencies between the stages of the algorithm. However, each stage can be implemented on the GPU fully parallel. The stages of the Systematic resampling method are given in Algorithm 2 [13].

Algorithm 2: Systematic Resampling [13]

```
[ $\{O^i,-,-\}_{i=1}^N$ ] = SYSTEMATIC [$\{\widetilde{w}^i\}_{i=1}^N$]
    $u \sim v[0\ 1)$
    C = INCLUSIVE-PREFIX-SUM($\widetilde{w}$)
    foreach $i = 1{:}N$
        $r^i = \frac{N*C^i}{C^N}$
        $O^i = \min(N, \lfloor r^i + u \rfloor)$
    end foreach
```

The output of this method is the cumulative summation of the number of repetitions of the particles called as cumulative offspring and denoted as O. C is the cumulative summation of the weights. u is a real random number between 0 and 1, excluding 1, drawn from a uniform distribution. r is a real number.

The iterations of 'foreach' loop can be executed in parallel among the threads where each thread runs for each particle. Each thread calculates the cumulative offspring of the particle it represents. We need to convert O to an ancestor array in order to assign the states of the ancestor of the particle. An algorithm about this conversion is given in Algorithm 3 [13]. It is very suitable for the GPU implementation.

Algorithm 3: Cumulative Offspring to Ancestors [13]

```
[ $\{xnew^i,-,-\}_{i=1}^N$ ] = CO TO ANCESTORS [$\{x^i,O^i\}_{i=1}^N$]
    foreach $i = 1{:}N$
        if i == 1  start = 0
        else  start = $O^{i-1}$
        end if
        $o^i = O^i - start$
        for $j = 1{:}o^i$
            $xnew^{start+j} = x^i$
        end for
    end foreach
```

In this algorithm, the iterations of 'foreach' loop can be executed in parallel among the threads where each thread runs for each particle. Each thread finds the children of the particle it represents if exists.

In our multi-pedestrian tracking problem, we have multiple pedestrians along with false alarms (measurements do not belong to any pedestrian). We need to identify measurements which ones belong to the pedestrians and which ones are false alarms. And we need to associate the measurements to the pedestrians correctly in order to track them persistently. There are algorithms to perform these issues. First, we mention some concepts about the multi pedestrian tracking problem. We track all measurements we receive from the sensor. A track can be in three status:

• Tentative track: Still undecided track whether it is target or false alarm

• Confirmed track: Track confirmed as target

• Deleted track: Absence of data of the track for a consecutive time steps. It can be tentative or confirmed track

To associate a measurement to a track, we use gating technique. We draw an ellipsoidal gate for a track by using predicted measurement error and its covariance and the state of the measurement. The measurements are located inside the gate of the track if their distances to the track are below than the gate threshold. If there are more than one measurements located inside the gate of the track or if more than one tracks have the same measurement inside their gates, we need a mechanism to associate the measurements to the tracks. We use nearest-neighbor (NN) technique. In this technique, the closest measurement to the track is assigned as the measurement of the track. The other ones are processed as new tentative track. If more than one tracks have the same measurement inside their gates, the oldest track has the priority to get the measurement. Once we associate all measurements with the existing tracks, we update the kinematic equations of the tracks by using the associated measurements. To delete a track we use M/N logic. We set M/N as 2/3. For the tentative tracks, they become confirmed if the measurements are associated with the track in 2 consecutive scans and are associated at least 2 of 3 following scans. Otherwise the track is deleted. For the confirmed tracks, they are deleted if the measurements are not associated with the track in 2 consecutive scans or are not associated at least 2 of 3 following scans. Note that nearest-neighbor is a hard decision mechanism and is suitable for the single target tracking problems. For the multi pedestrian tracking problems, we use global nearest neighbor (GNN) mechanism. In this mechanism, the measurements are associated to the tracks in the best hypothesis. Best hypothesis can be found with the assignment problem. We use auction algorithm for the assignment problem [14]. The pseudo-code of the multi-target tracking algorithm is given in Algorithm 4.

Algorithm 4 Multi-Pedestrian Tracking Algorithm

```
At time k = 0:
        Initialize tentative tracks
At time k > 0:
        If confirmed track exists:
                Gating(Confirmed Tracks, Measurements,GateResults)
                GNN(Confirmened Tracks, Measaurements, GateResults, AssociationResults)
                UpdateParticleFilter(Confirmened Tracks, Measaurements, AssociationResults)
                UpdateM/NLogic(Confirmened Track, AssociationResults)
        end if
        If tentative track exists, they are processed with the un-used measurements:
                Gating(Confirmed Tracks, Measurements,GateResults)
                GNN(Confirmened Tracks, Measaurements, GateResults, AssociationResults)
                UpdateParticleFilter(Confirmened Tracks, Measaurements, AssociationResults)
                UpdateM/NLogic(Confirmened Track, AssociationResults)
        end if

        Create new tentative tracks from the un-used measurements
```

In this algorithm, at time 0 we initialize the tentative tracks by using the measurements at time 0. These measurements can be either a target or a false alarm. At time $k > 0$, the measurements are associated with the confirmed tracks at first. Then the measurements that are not associated with the confirmed tracks are used for the tentative tracks in order to be associated with them. If there exists measurements that are not associated with the confirmed tracks and the tentative tracks, new tentative tracks are created with them. In Gating function, we calculate the distances of each measurement to the tracks. If the distance of the measurement is less than the gate threshold of the track, that measurement is considered to be inside of the gate of the track. In GNN function, we associate each measurement that is inside of the gates of the tracks with the tracks. In UpdateParticleFilter function, the particle filter of each track is processed by using the associated measurements. If there is no any associated measurement for a track, only the prediction stage of the particle filter is processed. In UpdateM/NLogic function, the parameters of the M/N logic are updated.

4. Experimental Environment and Results

We use SIR particle filter in our multi-pedestrian tracking problem. In the experiments, we use the real data of the pedestrians that are collected by [3]. They use multiple 3D range sensors in a shopping center to get the data of the pedestrians as the measurements of the problem. We select three specific person's data for our application. We apply a simple filter to eliminate the noises in order to get the true data of the persons. Then we add required noises to the true data to create our noisy measurement data. The units of the data in the tracking application are millimeter (mm) and seconds (sec.). The true data of the pedestrians are shown in Figure 1.

The arrows show the starting points of the pedestrians and the circles show the meeting points of the pedestrians or a sudden change (around $180°$) in the direction of the second pedestrian at a point. Starting time (Ts^1) of the first pedestrian is 5, starting time (Ts^2) of the second pedestrian is 292 and starting time (Ts^3) of the third pedestrian is 1433. The number of false alarms (clutters) is 10 at each time step. We add the measurement data of pedestrian 1, pedestrian 2 and pedestrian 3 to the clutter set before running the algorithm. Therefore, there exists at least 10 measurements and at most 13 measurements at each time step. The state of a pedestrian in the particle filter is $[p_x p_y v_x v_y]'$ where p_x is the x position and p_y is the y position of the pedestrian in 2D dimension. And v_x is the velocity in x position and v_y is the velocity in y position. The initial state of each particle is generated as follows:

$$\mathbf{x_0^i} \sim \mathcal{N}(\mathbf{x_0}, \mathbf{P_0}) \tag{1}$$

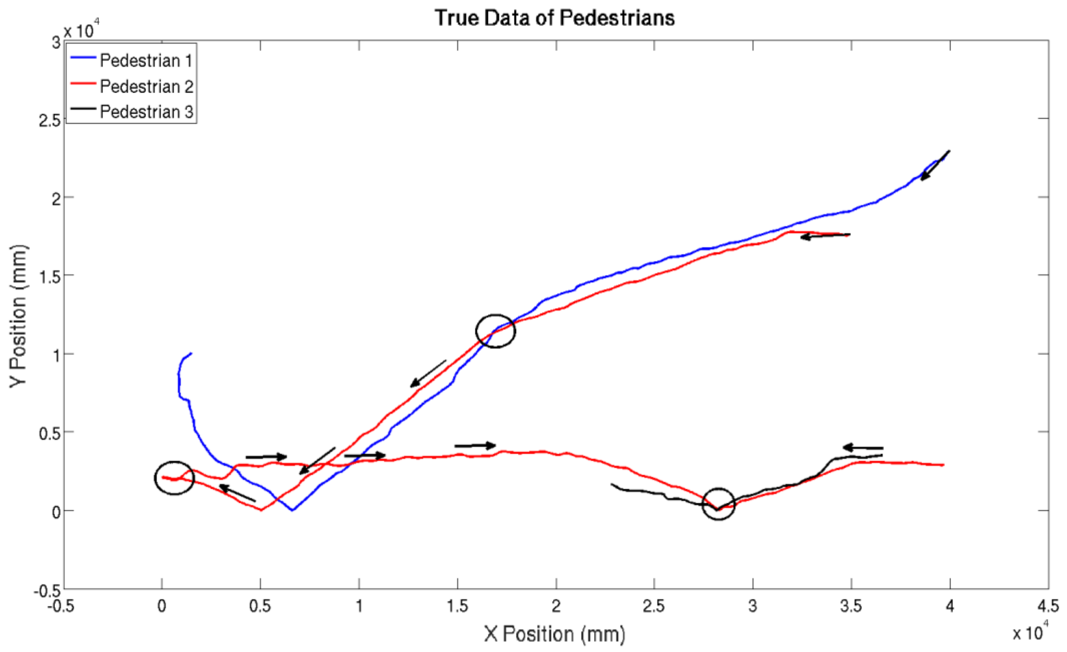

Figure 1 True data of the pedestrians we track in our application.

where x_0 is the initial estimation of the track and is equal to vector $[p_{x0} p_{y0} 0\ 0]'$ and P_0 is the multivariate process noise covariance matrix of x_0 and is equal to $diag[40^2\ 20^2\ 40^2\ 20^2]$. We use the nearly constant velocity model in the prediction process of the particle filter and the formula of the prediction process of a particle is defined as follows:

$$\mathbf{x_k^i} = \begin{bmatrix} \mathbf{1\ 0\ T\ 0} \\ \mathbf{0\ 1\ 0\ T} \\ \mathbf{0\ 0\ 1\ 0} \\ \mathbf{0\ 0\ 0\ 1} \end{bmatrix} * \mathbf{x_{k-1}^i} + \begin{bmatrix} \frac{\mathbf{T^2}}{\mathbf{2}} & \mathbf{0} \\ \mathbf{0} & \frac{\mathbf{T^2}}{\mathbf{2}} \\ \mathbf{T} & \mathbf{0} \\ \mathbf{0} & \mathbf{T} \end{bmatrix} * \mathbf{q_k^i} \tag{2}$$

where T is the sampling period and $q_k^i \sim \mathcal{N}(0, Q)$ is the vector of the multivariate normally distributed random numbers. Q is the multivariate process noise covariance matrix. We set the value of Q as $diag[1500\ 1500]$ by considering the average velocities in x and y positions of the pedestrians and set the value of T as 1 since the duration between the two consecutive measurements is one second. The measurement data are the $range$ and the $bearing$ of the track and are defined as follows:

$$\mathbf{z_k} = \begin{bmatrix} \sqrt{\mathbf{p_{xk}^2} + \mathbf{p_{yk}^2}} \\ \mathbf{arctan}\,(\mathbf{p_{yk}}/\mathbf{p_{xk}}) \end{bmatrix} + \mathbf{v_k} \tag{3}$$

where $v_k \sim \mathcal{N}(0, R)$ is the vector of the multivariate normally distributed random numbers and R is the multivariate measurement noise covariance matrix. We set the value of R as $diag[40^2 \left(\frac{0.1pi}{180}\right)^2]$ by considering the noises in measurements. The weight of a particle is calculated by applying the state of the particle x_k^i to the multivariate normal probability density function of the measurement whose mean is the measurement data z_k and covariance is R.

We obtain the estimation of the track as the combination of the particles according to their normalized weights. The computation is done as follows:

$$\hat{\mathbf{x}}_{\mathbf{k}} = \sum_{\mathbf{i}=1}^{\mathbf{N}} \mathbf{x}_{\mathbf{k}}^{\mathbf{i}} \mathbf{w}_{\mathbf{k}}^{\mathbf{i}} \tag{4}$$

In resampling, we implement the parallel Systematic resampling algorithm given in Section 3 on the GPU. We use an efficient parallel reduction algorithm for the sum operations of the SIR particle filter on the GPU and parallel scan algorithm for the inclusive scan operation of the Systematic resampling method [15][16]. To assess the quality of the tracking of the pedestrians, we calculate the root mean squared error (RMSE) of the estimation of the pedestrians. The estimation error in a time step is defined as follows:

$$\mathbf{e}_{\mathbf{k}}^{\mathbf{j}} = \sqrt{\left(\mathbf{p}_{\mathbf{xk}}^{\mathbf{j}} - \mathbf{xk}_{\mathbf{1}}^{\mathbf{j}}\right)^2 + \left(\mathbf{p}_{\mathbf{yk}}^{\mathbf{j}} - \mathbf{xk}_{\mathbf{2}}^{\mathbf{j}}\right)^2} \tag{5}$$

where xk_1^j and xk_2^j are the estimated positions of jth pedestrian in x position and y position respectively at time step k. p_{xk}^j and p_{yk}^j are the true positions of jth pedestrian in x position and y position respectively at time step k. The RMSE is calculated with the estimation error e_k^j at each time step k. It is defined follows:

$$\mathbf{RMSE}^{\mathbf{j}} = \sqrt{\frac{\sum_{\mathbf{k}=1}^{\mathbf{Tst^j}} \left(\mathbf{e}_{\mathbf{k}}^{\mathbf{j}}\right)^2}{\mathbf{Tst^j}}} \tag{6}$$

where Tst^j is the total time step of jth pedestrian.

We implement the serial multi pedestrian tracking algorithm by using C++ programming language on Intel Core i7-4790K CPU [17]. We implement the parallel multi-pedestrian tracking algorithm by using CUDA 7.5 programming environment on NVIDIA Tesla K40 board [18][19].

The execution time results of the serial and the parallel multi-pedestrian tracking algorithms are given in Table 1 and 2 along with the speed up results.

Table 1: Total Execution Time Results of Multi Pedestrian Tracking Application (in seconds)
(The number of total time step is 1900)

$\log_2 N$	11	12	13	14	15	16	17	18	19	20
C++	24.19	57.60	93.37	191.02	457.99	761.09	1535.62	3246.65	6390.78	11829.70
CUDA	21.56	31.32	36.90	47.24	80.08	108.56	201.20	382.38	707,94	1207.47
Speed Up	1.12x	1.83x	2.53x	4.04x	5.71x	7.01x	7.63x	8.49x	9,02x	9.79x

Table 2: Average Execution Time Results with respect to the time step (in seconds)
(The average duration of one time step)

$\log_2 N$	11	12	13	14	15	16	17	18	19	20
C++	0,012	0,030	0,049	0,10	0,24	0,40	0,80	1,70	3,36	6,22
CUDA	0,011	0,016	0,019	0,024	0,042	0,057	0,10	0,20	0,37	0,63

The results show us that as the number of particles increases, the execution time of the multi pedestrian tracking application which is implemented on the GPU improves. And we achieve up to 9.79x speed up. When we investigate the average execution times, the improvement on the execution time of the multi pedestrian tracking application with CUDA implementation becomes substantial. When we consider the duration between two consecutive measurements is less than one second, the performance of the CUDA implementation provides us an important solution as the number of particles becomes very large. The RMSE results of three pedestrians are given in Table 3, 4 and 5.

Table 3: RMSE Results of Pedestrian 1

$\log_2 N$	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
C++	34.07	32.92	36.79	32.91	32.94	32.92	32.07	32.09	32.93	33.01
CUDA	33.58	32.05	32.14	32.94	32.13	32.92	32.09	32.08	32.08	32.08

Table 4: RMSE Results of Pedestrian 2

$\log_2 N$	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
C++	37.27	37.43	38.54	36.13	36.95	37.58	35.31	35.88	36.16	38.85
CUDA	37.96	37.05	36.43	35.92	35.57	36.37	35.94	35.59	35.00	35.47

Table 5: RMSE Results of Pedestrian 3

$\log_2 N$	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
C++	36.59	37.20	38.26	40.46	38.59	39.49	39.32	38.03	39.60	41.88
CUDA	34.72	37.01	33.98	34.89	44.48	38.76	35.75	38.64	49.77	53.49

It is seen that the RMSE results of three pedestrians are around 3-4 cm which can be considered as very low error. Further, the RMSE results of C++ and CUDA implementations are very close to each other. So we can choose the CUDA implementation when the speed is very important in the multi pedestrian tracking problem.

5. Conclusion

When there are too many pedestrians in an area and the particle filter is necessary for the multi pedestrian tracking problem, serial implementation of the particle filter has high complexity in terms of execution time and can not be enough for the applications where the duration between two consecutive measurements is small. Due to have many cores in their architectures, GPUs offer promising solutions for the problems which need the particle filter as the estimation method. In a multi pedestrian tracking application, there may be too many pedestrians along with false alarms (clutters). In our application, we receive at most 13 measurements in a time step from the sensor which means at most 13 particle filters run at a time step. We try to implement the stages of the particle filters on the GPU in parallel and achieve up to 9.79x speed up. In the future, we can extend our multi pedestrian tracking problem by adding new pedestrians to the problem. We can also implement the problem on a multi GPU platform where each particle filter of the measurements runs in different GPU.

Acknowledgments

We gratefully acknowledge the support of NVIDIA Corporation with the donation of Tesla K40 GPU used in this work. This work was also supported in part by Republic of Turkey Ministry of Development, Turkey under grant BAP-08-11-DPT2002K120510. Furthermore, the author was supported in part by The Scientific and Technological Research Council of Turkey (TUBITAK) under program 2214/A. Thanks are due to Prof. Fusun Ozguner for her support for the author during his visit to Ohio State.

References

[1] B. Ristic, S. Arulampalam, and N. Gordon, *Beyond the Kalman Filter Particle Filters for Tracking Applications.* Artech House, 2004.

[2] P. Gong, J. D. Basciftci, and F. Ozguner, "A Parallel Resampling Algorithm for Particle Filtering on Shared-Memory

Architectures," *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International, Shanghai*, pp. 1477-1483, 2012.

[3] D. Brscic, T. Kanda, T. Ikeda, and T. Miyashita, "Person position and body direction tracking in large public spaces using 3D range sensors," *IEEE Transactions on Human-Machine Systems*, Vol. 43, No. 6, pp. 522-534, 2013.

[4] G. Hendeby, J. D. Hol, R. Karlsson, and F. Gustafsson, "A graphics processing unit implementation of the particle filter," *In Signal Processing Conference, 2007 15th European*, pp. 1639–1643, IEEE, 2007.

[5] K. Hwang, and W. Sung, "Load balanced resampling for real-time particle filtering on graphics processing units," *IEEE Transactions on Signal Processing*, Vol. 61, No. 2, pp. 411–419, 2013.

[6] Y. Wu, J. Wang, and Y. Cao, "Particle filter based on iterated importance density function and parallel resampling," *Journal of Central South University*, Vol. 22, No. 9, pp. 3427–3439, 2015.

[7] M. Owczarek, P. Barański, and P. Strumiłło, "Pedestrian tracking in video sequences: a particle filtering approach," *In Computer Science and Information Systems (FedCSIS), 2015 Federated Conference on*, pp. 875-881, 2015.

[8] Y. Guan, X. Chen, Y. Wu, and D. Yang, "An improved particle filter approach for real-time pedestrian tracking in surveillance video," *In International Conference on Information Science and Technology Applications,* Atlantis Press, 2013.

[9] D. Stadler, and J. Beyerer, "Improving Multiple Pedestrian Tracking by Track Management and Occlusion Handling," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[10] F. Flodin, *Improved Data Association for Multi-Pedestrian Tracking Using Image Information*, Master of Science Thesis in Electrical Engineering Department of Electrical Engineering, Linköping University, 2020.

[11] L. Barba-Guaman, J. Eugenio Naranjo, and A. Ortiz, "Deep learning framework for vehicle and pedestrian detection in rural roads on an embedded gpu," *Electronics*, Vol. 9, No. 4, pp. 589, 2020.

[12] M. Dimitrievski, P. Veelaert, and W. Philips, "Behavioral pedestrian tracking using a camera and lidar sensors on a moving vehicle," *Sensors*, Vol. 19, No. 2, pp. 391, 2019.

[13] L. M. Murray, A. Lee, and P. E. Jacob, "Parallel resampling in the particle filter," *Journal of Computational and Graphical Statistics*, Vol. 25, No. 3, pp. 789–805, 2016.

[14] S. Blackman, and R. Popoli, *Design and Analysis of Modern Tracking Systems*. Norwood, MA: Artech House, 1999.

[15] M. Harris, "Optimizing Parallel Reduction in CUDA, NVIDIA Developer Technology," 2007. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf

[16] W. M. Hwu, "A work-eficient parallel scan kernel," 2014. [Online]. Available: http://ece408.hwu-server2.crhc.illinois.edu/Shared%20Documents/Slides/Lecture-4-6-work-efficient-scan-kernel.pdf

[17] Intel, "Intel® Core™ i7-4790K Processor," 2020. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/80807/intel-core-i7-4790k-processor-8m-cache-up-to-4-40-ghz.html

[18] NVIDIA, "Tesla K40 GPU Active Accelerator: Board Specification," 2013. [Online]. Available: https://www.nvidia.com/content/PDF/kepler/Tesla-K40-Active-Board-Spec-BD-06949-001_v03.pdf

[19] NVIDIA, "NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110/210," 2014. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf