Particle Filter for Video-based Object Tracking

Ramkiran Balivada, Karan Jeswani
Department of Computational and Data Science
Indian Institute of Science, Bangalore, India
ramkiranb@iisc.ac.in, karanjeswani@iisc.ac.in

***Abstract*—We propose a new method for Parallel Resampling, and implement video-based parallel object tracking using particle filter algorithm which is based on recursive Baye's state estimation technique. We parallelized dynamics and measurement models and Systematic resampling method. The object tracking was performed with input video using opencv and CUDA softwares.**

I. Introduction

A. Particle Filtering in Video Tracking

Particle Filter method is a recursive Bayesian Monte Carlo method used to solve the filtering problems. The filtering problem that we are interested in is to track a moving object in a video.

A recursive Bayesian filtering procedure is used to compute the posterior distribution of a the state of a system, given the initial conditions and a measurement. In the current setup, initial selected image's position forms the initial condition while its position in subsequent frames forms measurements.

Recursive Bayesian filtering method generally comes with many different flavours, namely Kalman Filter, extended Kalman Filter, unscented Kalman Filter, Particle Filter etc. We have specifically selected Particle Filter as it can address the non-linearities present in the measurement model more accurately than any other above mentioned filters. Particle Filters can also estimate the posterior distribution even under non-gaussian processes.

It does this by representing a distribution using many samples taken from the distribution. This is called the non parametric representation of a state. Each particle then represents the possible states that the system could be in at the next time step.

Then we assign weights to each individual particle based on a measurement. These weights represent the likelihood of the particles being in the true state.

In Particle Filter, the posterior statistics of the estimate is approximated by the distribution formed by the particles at that instant.

Particle Filter estimation technique involves the following steps:

1) **Prediction** - Use system model to propagate the state of the system as a new prediction.
2) **Update** - Based on measurement data, we find the new estimate of the state.
3) **Resampling** - Is performed to avoid weight degeneracy

B. Parallelization of Particle Filter

The process of estimation using Particle Filters requires 3 major steps as mentioned above. In the prediction step, each particle is propagated using a forward model and since these operations are independent of other particles, these operations can be easily parallelized.

Also in the update step, each particle likelyhood is computed based on the latest measurement and its current state resulting in easy parallelization.

The resampling step is inherently sequential in nature. A parallelized resampling method is addressed in Nicely et al. [1]. In this work, we tried a binary search based approach for resampling. This approach is explained in section III.

II. Related Work

The parallel resampling method in Nicely et al. [1] is based on linear search with mid-point initialization. We learn the method and implemented the parallel resampling algorithm.

The work by Abdelali et al. [2] implements a video based object detection using particle filter.

We combine the work of both of the papers, and implement a parallel object detection using particle filter.

III. Methodology

A. Video-based object tracking using Particle Filter

Prediction Step: In prediction step, we predict the state of each particle after propagating through a dynamics model. This dynamics model describes the motion of the moving object in the video. In the current setup, we use a linear system model that describes motion of particles. Initially, we allow a large variance in state estimates so that we get a good spread of particles across the image. This high variance ensures to capture all possible states of object position.

Update step:
Update step involves in updating the state estimate as we get newer measurements. The measurement model used in the current scenario is a color histogram.
We measure color histogram from the image in the current time step at each particle position (which we call particle image) and compute the similarity of this color histogram with the target histogram (user selected region of the image, which we will call as target image). Particles with higher similarity measure are given higher weights, and vice-versa. Bhattacharyya co-efficient is used to find the similarity

between two histograms.

$$BC(p,q) = \sum_{i=0}^{B} \sqrt{P_i Q_i}$$

Where P_i is the target color histogram.
Q_i is the particle color histogram.
B is the histogram size.

Resampling Step: Resampling method samples more particles with higher weights and less particles with low weight. These newly sampled particles will be used in the next frame to continue the estimation process.
Systematic resampling method is used for resampling in our work.
Systematic resampling method uses Roulette table method to obtain the new samples from the updated weights.
In the roulette table method, we form a circle and its corresponding sectors (weight sectors) where the sector angles are proportional to the weights of the particles.
We need to sample N particles from the distribution represented by the updated weights. To do this, we divide the above mentioned circle into N equal parts (uniform sectors). The number of particles coming from a particular weight is equal to count of the number of uniform sectors in a given weight sector formed by that weight.

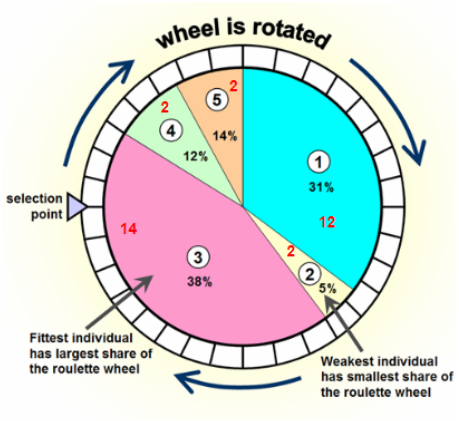


Fig. 1. Roulette Table description

In figure-1, the number of uniform sectors in each weight sector are highlighted in red. One can clearly see that higher weight particles get resampled more often (more intersections by uniform sectors).
In figure-2, we describe the estimation process used for object tracking using particle filter.

B. Algorithm for Resampling

In this section, we concentrate on the implementation of Systematic Resampling algorithm which forms the basis of our estimation process.

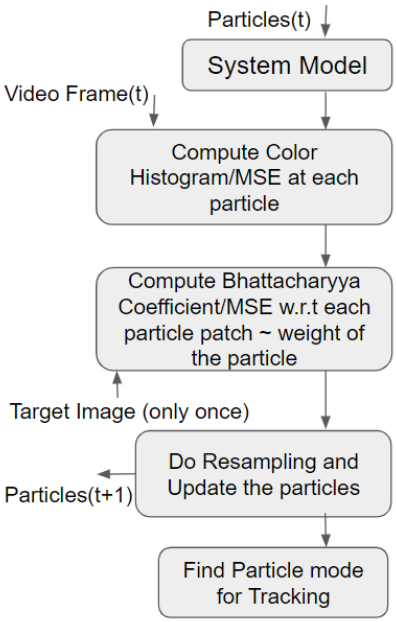


Fig. 2. Flowchart

Algorithm 1 is a serial implementation of the resampling algorithm. The resampling algorithm returns a new set of indices as per the weights of the particles. It chooses particles which have highest weights and removes particles which have lesser weights as described in roulette table method in IIIA.
In this method we input the cumulative distribution function (cdf) formed by the weights of the particles. This cdf is used to form the weighted sectors which will be subsequently used to count the number of indices representing a given particle. We take cdf as the input for the resampling algorithm as it is parallelized separately for object detection.
One can see that the problem of resampling boils down to finding the pair of indices in the cdf between which the uniform random number belongs and this process is repeated for all particle weights.

Algorithm 2 describes the parallel implementation of the resampling method used by Nicely et al. [1]. In this method each thread processes individual particle. Each thread initializes the index as per the thread index and finds the resampled index by performing a linear search over the cdf, in either direction.

Algorithm 3 is also another parallel implementation of the resampling algorithm which we propose in this work as an improvement over Algorithm 2. In this method, we use a binary search on the cdf as we indentify that cdf is a monotonically increasing array. Also since cdf is a large array, this will reduce the memory accesses from the global

memory in GPU.

Algorithm 1: Systematic Resampling (Serial)

Input: cdf, N, $u_0 \leftarrow$ cdf of particle weights, Number of particles, Random number
Output: new_idx $\leftarrow$ Resampled index

```
1 u = u_0 / N;
2 k ← 0;
3 for i ← 1, N do
4 |   while cdf[icdf] < u do
5 |   |   ++icdf
6 |   end
7 |   new_idx[i] = icdf;
8 |   u += (1.0 / N);
9 end
```

C. Parallel Optimizations

1) Particles array has 4 elements, which are position x, position y, velocity x, velocity y. These elements are stored in a contiguous memory so that they are fetched at once, thus improving memory access performance during CUDA kernel operations.
2) While finding the similarity between particle image and target image, we need to send all the particle images to the GPU, which involves lot of memory and communication. So instead, we have sent the entire frame to the GPU and selected particle images from the global memory.
3) Our method of finding prefix sum is that we first find prefix sum for each block (A), then find the prefix sum of the last elements (B) of each block prefix sum array. Then we do a vector add of block prefix sum array (A) and the newly formed last element prefix sum (B) to the previous elements in A.

IV. Experiments and Results

A. Experiment Setup

Compared serial resampling algorithm with 2 different parallel resampling algorithms which are:

1) Parallel Resampling using linear search with mid-point initialization [1].
2) Parallel Resampling using binary search (novel)

The implementation of all resampling algorithms is run for 2^{10} to 2^{20} number of particles to observe scaling due to parallelization in both algorithms.
Compare serial and parallel implementation of object tracking with different number of particles ranging from 2^{12} to 2^{16}.
Timing is obtained for :

1) Dynamics and measurement
2) Prefix sum
3) Systematic Resampling (roulette table resampling)

All 3 different phases of the particle filter are timed over all frames from input video and average time for each phase is reported.

Algorithm 2: Systematic Resampling (Parallel) in [1]

Input: cdf, N, $u_0 \leftarrow$ cdf of particle weights, Number of particles, Random number
Output: new_idx $\leftarrow$ Resampled index

```
1  t = blockIdx.x * blockDim.x + threadIdx.x;
2  idx = t;
3  u = (t + u_0)/N;
4  m = 1;
5  el = 0;
6  while m do
7  |   if t ≥ (N - el) then
8  |   |   m = 0;
9  |   else
10 |   |   m = (cdfd[t + el] < u);
11 |   end
12 |   if m then
13 |   |   idx++;
14 |   else
15 |   |   el++;
16 |   end
17 |   el = 1;
18 |   while not m do
19 |   |   if t ≤ el then
20 |   |   |   m = 1;
21 |   |   else
22 |   |   |   m = (cdfd[t - el] < u);
23 |   |   end
24 |   |   if not m then
25 |   |   |   idx–;
26 |   |   else
27 |   |   |   el++;
28 |   |   end
29 |   end
30 |   new_idxd[t] = idx;
31 end
```

B. Results

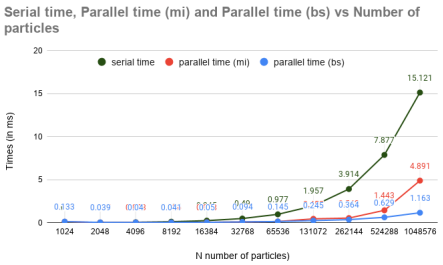
Figure-3 shows comparision of timing and speedup for both the parallel resampling algorithms. We can clearly see that binary search based parallel resampling is scaling better than the linear search method as the number of particles increases. This is because number of memory accesses in binary search based methods are less as compared to linear search based method, even though it has better initialization of the index.
Figure-4 shows timing performance of various phases during object tracking. Dynamics and measurement take most of the time in the serial program, and these are embarrassing parallel so we observe maximum improvement for them.
Also the Roulette Table Resampling takes the second highest time in the serial program. We see considerable speedup for this as seen in figure-5, which keeps on scaling with number of particles.

Algorithm 3: Systematic Resampling (Parallel) proposed

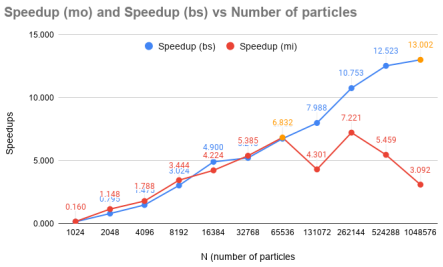
Input: cdf, N, $u_0 \leftarrow$ cdf of particle weights, Number of particles, Uniform Random number between 0 and 1

Output: new_idx $\leftarrow$ Resampled index

```
t = blockIdx.x * blockDim.x + threadIdx.x;
u = (t + u_0)/N;
m, l = 0, r = N - 1;;
while true do
    m ← (l + r) / 2;
    if cdfd[m] < u && cdfd[m+1] ≥ u then
        break;
    else if cdfd[m+1] < u then
        l ← m + 1;
    else if cdfd[m] ≥ u then
        r ← m;
    else
        // do nothing
    end
    if r ≤ l then
        m ← m - 1;
        break;
    end
end
new_idx[t] = m + 1;
```



(a) Times for both resampling algorithms



(b) Speedup for both resampling algorithms

Fig. 3. Time plot speed up plot wrt number of particles

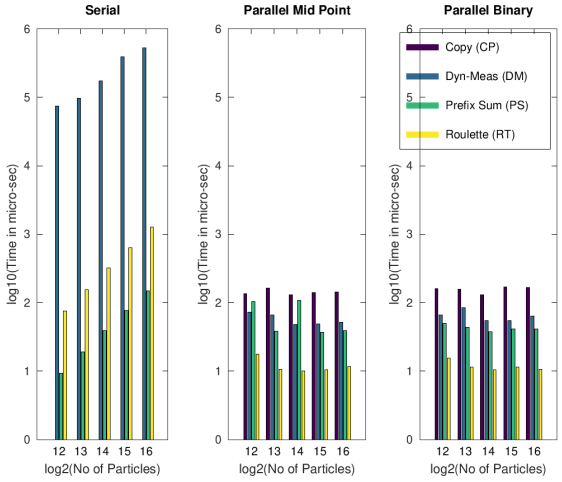


Fig. 4. Time contribution from different phases of the parallel program

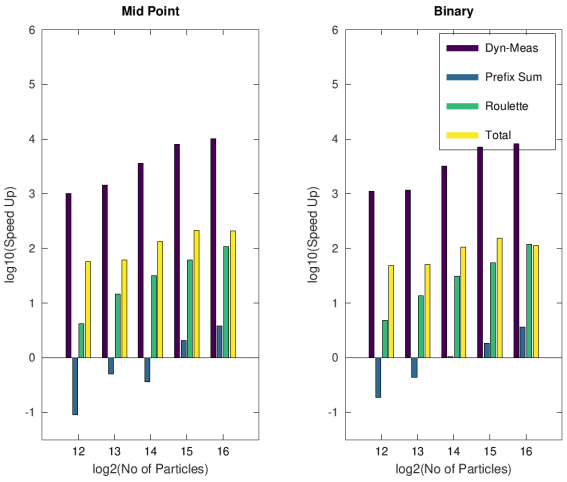
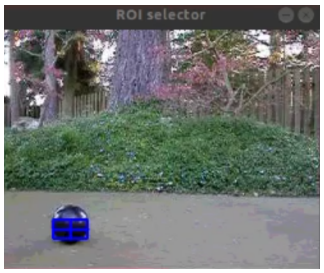


Fig. 5. Speedup comparision between 2 parallel algorithms

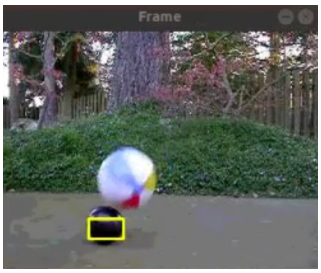
V. Conclusions and Future Work

A. Conclusions

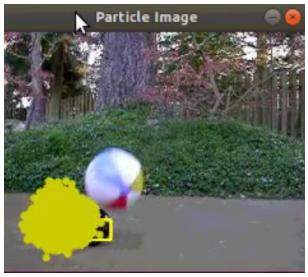
Our main contribution is parallelization of the resampling algorithm using binary search instead of linear search and developing object tracking framework.For parallel resampling we were able to get improvement over Nicely et al. [1] which gave speedup of 6.832 with speedup of 13.002. We observe better speedup for binary search based parallel resampling algorithm. But the times obtained were still not better than Nicely et al. [1] because they have used cache performance and some other optimizations (like using local memory) that we did not have time to do. We successfully implemented parallel object detection, in video.



(a) user selected region



(b) Speedup for both resampling algorithms



(c) Bounding box tracking black ball

Fig. 6. Particle distribution around bounding box

B. Future Work

Parallel particle filter can be used for multiple target tracking. We can use different system model for prediction step based on knowledge of motion of object. If we have limited knowledge of the object motion, we can still use high variance in linear model to ensure proper tracking. With proper memory management (using shared memory), we reduce global memory accesses thus further improving speedups.

REFERENCES

[1] M. A. Nicely and B. E. Wells, "Improved parallel resampling methods for particle filtering," *IEEE Access*, vol. 7, pp. 47 593–47 604, 2019.

[2] H. Abdelali, F. Essannouni, and D. Aboutajdine, "Object tracking in video via particle filter," *International Journal of Intelligent Engineering Informatics*, vol. 4, p. 340, 01 2016.