GAMIFICATION OF PARTICLE FILTER BASED MULTI-OBJECT TRACKING THROUGH SOCKET BASED COMMUNICATION

ABSTRACT

Inspired by the popular series, 'Squid Games', we create a gamified framework that tracks players' movements and checks for their adherence to instructions through master-client communication. We employ particle-filter based muli-object tracking, robust enough to withstand occlusion, and evaluate our performance with and without multi-threading. The tracking output is communicated via single server-multiple client environment using socket programming. In addition, we have also explored GPU programming to accelerate the particle filter algorithm.

1 Introduction

1.1 Background

Game design The participants are sent to a room with a camera in front of them. The participants will follow instructions to go left or right in the room. There is a master server which takes as input the paths to be followed by each participant. The camera in the room tracks the participant using a particle filter algorithm and then sends back position data to the master. The master uses this position data to determine whether the participants have followed the rules correctly or not which will determine whether the participant will be eliminated or allowed to move on to the next round.

Object tracking and detection The aim of a target tracking algorithm is to estimate the target position precisely from the partial noisy observations available [1]. Object detection can be carried out with a variety of techniques like template matching, Hough transforms etc. In object detection, we detect the object independently in each frame while recording it's position over time. Image tracking, in addition to image measurements, also incorporates the expected position pattern of the object. Here we rely on the expected position of the object to predict it's position before the next frame even occurs. Tracking can be seen as the process of propagating the posterior distribution of state given measurements across time. Kalman Filters are simple and compact but carry the disadvantage of being too sensitive to noise and rely on a single hypothesis.

Particle Filter and Bayesian methods Particle filters were first introduced to the computer vision community by the famous condensation paper of Isard and Blake [2] for contour tracking.Particle filters use multiple discrete "particles" to represent the belief distribution over the location of a tracked target [3]. In order to understand the working of particle filters, we need to start at Bayesian Filters. Bayesian filtering refers to estimating the state of a dynamical system using a predict/update cycle based on sensor measurements. In this paradigm, we continuously update the posterior distribution based on the current state X_t given the observations Z_t. This forms the idea to represent the posterior density by a set of random particles with associated weights and compute estimates based on these samples and weights P(X$_t|Z^t$) $= kP(Z_t|X_t)P(X_t|Z^{t-1}) = kP(Z_t|X_t) \int_{X_{t-1}} P(X_t|X_{t-1})P(X_{t-1}|Z^{t-1})$.

2 Methodology

The task of gamification consists of inter-play of different components that will be described in the section.Each component is built using the concepts and skills learnt in ECE 6122. The setup consists of two participants in each room being tracked by a camera present in the room.

- Each participant is instructed to follow the master's instructions as they enter a room.

- The master relay's the instruction to the player at run-time, namely, "GO LEFT", "GO RIGHT"
- Particle Filter based object tracking determines player's adherence to rules through the co-ordinates returned by the camera.
- Making the wrong turn lead's to the participant's elimination from the game.
- Winners from each room proceed to the next level.

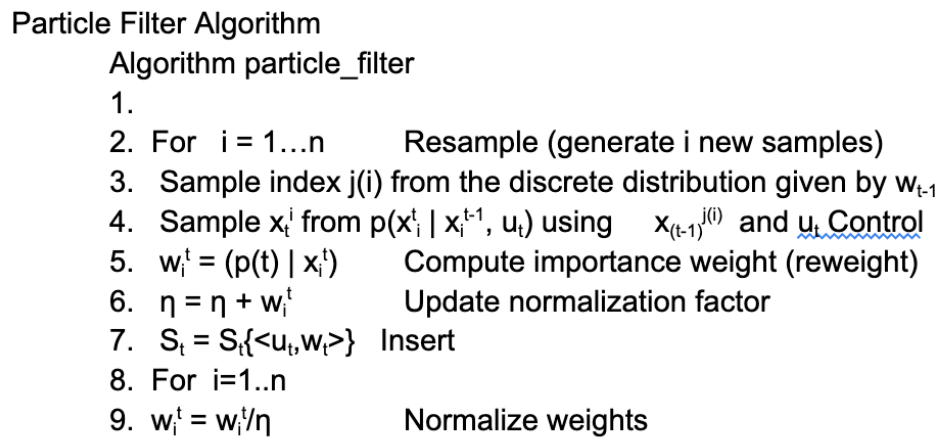
2.1 Pre-processing

The video consisting of the participants' motion is split into frames using OpenCV. These frames serve as input to the particle filter. We fine-tune the tracking process by providing the particle filter algorithm the exact set of frames where tracking starts.

2.2 Particle Filter

The main idea behind the particle filter is to sample a set of particles where each particle represents how likely the object is expected to be present in the given sample space. Once all the particles are sampled and evaluated, a weight is assigned to them based on their correctness. All the particles with a higher weight are kept whereas the lower weighted particles are removed from the next iteration of re-sampling. This process is repeated continuously for each frame.

In the implementation, the particle filter takes a template(for the person to be tracked) and parameters like the start frame, end frame for tracking, number of particles, standard deviation for the particles and other hyper-parameters as input. It is also given the frames obtained from pre-processing. The tracking happens, sequentially, frame-to-frame, with particle weight recomputed in each frame to get an estimate of the location of the object as (x, y) co-ordinates.



```
Particle Filter Algorithm
    Algorithm particle_filter
    1.
    2. For   i = 1…n          Resample (generate i new samples)
    3.   Sample index j(i) from the discrete distribution given by $w_{t-1}$
    4.   Sample $x_t^i$ from $p(x_i^t \mid x_i^{t-1}, u_t)$ using   $x_{(t-1)}^{j(i)}$ and $u_t$ Control
    5.   $w_i^t = (p(t) \mid x_i^t)$        Compute importance weight (reweight)
    6.   $\eta = \eta + w_i^t$             Update normalization factor
    7.   $S_t = S_t\{<u_t,w_t>\}$   Insert
    8. For  i=1..n
    9. $w_i^t = w_i^t/\eta$               Normalize weights
```

Figure 1: Code walkthrough

2.3 Multi-threading tracking

The core-tracking logic as described in the previous section is multi-threaded. Each thread tracks one person. The reasoning for this optimization is that tracking of participant A can happen independently of tracking of participant B. More concretely, the weight computation for the particles of a participant(which determine the location of a participant) can happen independently of that of the weight computation of the other participant. Python's threading library was used for this task.

2.4 Socket communication

As the camera tracks the participants, it relays the co-ordinates information to the master, which is a server which stores this information for decision-making. The master acts as a server: it has a socket which binds on a port, which is known to the camera node. The master also tells the cameras the participant IDs it needs to track so that the camera can tag the information accordingly when sending co-ordinate information. TCP sockets are used to maintain a connection-oriented setup between the master and the cameras. Both threads of the camera(tracking different participants) use the same connection to communicate to the master. Python's sockets and threading library was used for this task.

2.5 Object-oriented Setup

The master's functionality is contained in the server class, the camera's client and tracking functionality is contained in the client class. This design enables us to spin up cameras as objects, without repeating code. This is adhering to the Don't Repeat Yourself(DRY) principle.

Algorithm :

master :
1) Create server socket, bind to port, wait for clients
2) Once a client connects, create a thread for this communication
3) In each thread, receive, parse and store the co-ordinate data from the clients.
4) Once all clients are done tracking, review stored data and decide which participant needs to be eliminated.
5) Display results in the form of graphs and tables.
6) Terminate.

client :
1) Connect to server.
2) Get list of partipant Ids from the server. Create a thread per participant for particle filter , pass participant ID and template as arguments, start the threads.
    1. In every frame, compute the (x, y) for the participant
    2. Use the socket created in step 1. to send data to the server.
3) Wait for threads to join. Send message to server that tracking is complete.
4) Render tracking
5) Terminate

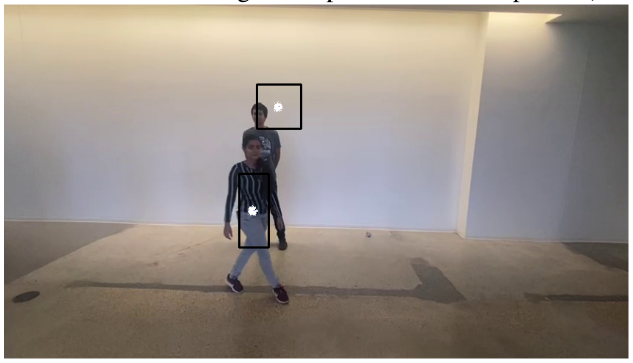
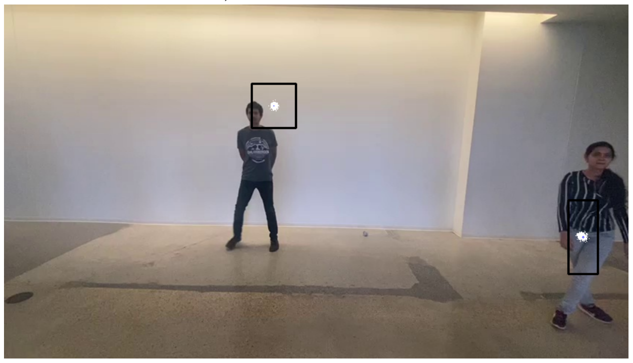
Figure 1-4Project workflow

2.6 Post Processing

This section uses the position data obtained from the camera in order to generate the kill list and output plots. This involves CSV file handling.

3 Results

Particle Filter Tracking Participant 0 and Participant 1 (taken at different frames)

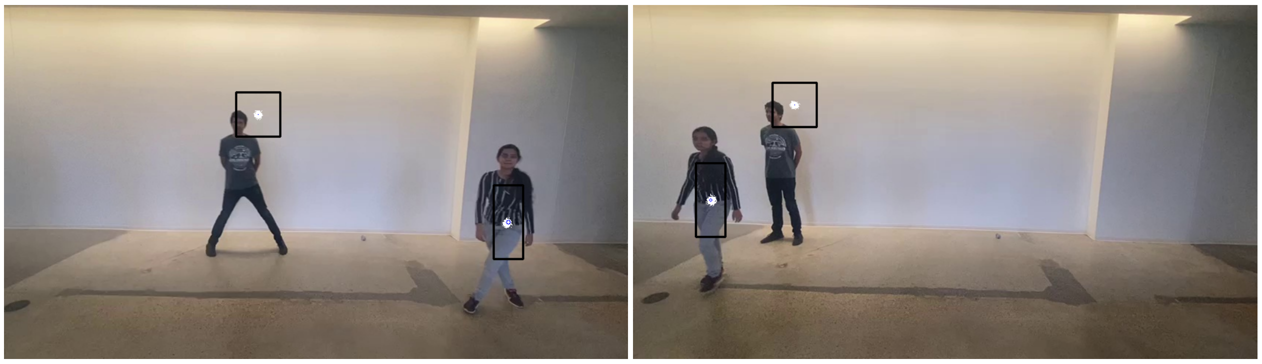


Figure 1-5: Participants tracked at different locations. The black bounding box around each participant (object) is the region where they are expected to be present and the white cluster of particles are the weighted center. If the cluster is very spread out, it means that the tracked location has lower confidence. In the above images, we can observe that the particles are fairly concentrated at the center indicating a good estimate of tracking.

Path trajectory plot for Participant 0 and Participant 1.

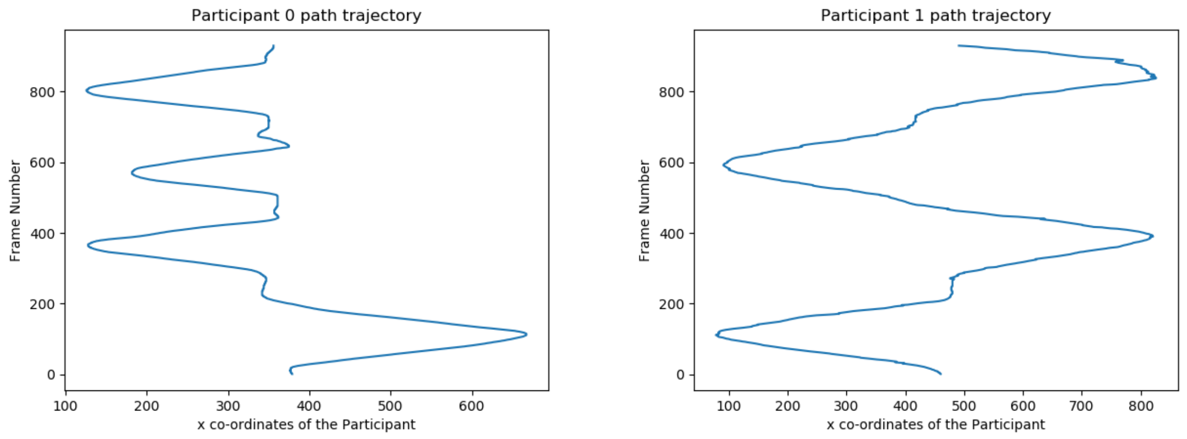
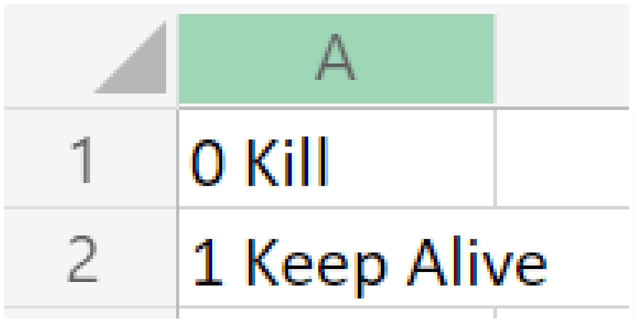
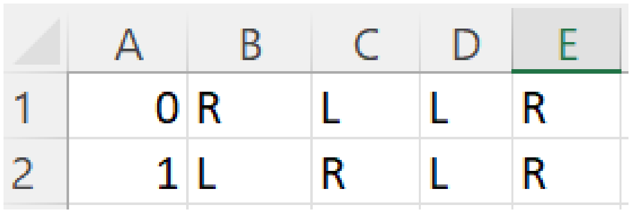


Figure 1-2: As observed from the first video, Participant 0 moves in the following pattern -> (Right - Left - Left - Left) whereas Participant 1 moves in the following pattern -> (Left - Right - Left - Right). This path is reflected in the above plots where y axis represents the frame number and the x axis represents the x-coordinate of the participant's position at that frame.



	A
1	0 Kill
2	1 Keep Alive



	A	B	C	D	E
1	0	R	L	L	R
2	1	L	R	L	R

Figure 1 - The kill-list based on the obtained data from the clients and Figure 2 - The Path set by the server for each participant : The decision is made for each participant based on whether or not they followed the path defined by the server. Eg: For Participant 0, The prescribed path is Right-Left-Left-Right but the Participant 0 moves Right-Left-Left-Left in reality, hence deviating the path. The result is 'kill'. However, the Participant 1 follows the prescribed path resulting in a decision, 'Keep Alive' and will progress to the next round, if any.

3.1 GPU Acceleration for Particle filter

In order to test if GPU parallelization is possible for the Particle filter algorithm, we tested the algorithm on a single object in a custom shorter video. We have provided a detailed view of the testing here. We implemented GPU acceleration using a package called numba which uses CUDA plugins to help create vectorized code and run it on the GPU. We observed that it executed for 15.9 sec with GPU acceleration and about 18.1 sec with CUDA disabled.

4 Challenges

In this section, we describe some of the challenges faced during the ideation and implementation of the project.

- Tracking using particle filters requires hyper-parameter tuning to achieve decent accuracy; and this needs to be done for every new video.
- CUDA and GPU programming requires us to find sections of code which can be parallelized and then need to re-implement them in a way that it can be executed on the GPU to enhance performance. This task was very challenging when it came to parallelizing a complex and computationally intensive algorithm like Particle filter.

5 Resources

[1] Github repository: https://github.com/vikasrao7/APT-Project [2] Python Sockets documentation [3] Python Threading documentation

6 Work distribution

Particle filter implementation	Vikas	Rahul
Sockets Implementation	Sahana	Vikas
Multi-threading	Rahul	Adithya
GPU Acceleration	Adithya	Vima
Video and Report	Vima	Sahana

References

[1] Rooji Jinan and Tara Raveendran. Particle filters for multiple target tracking. *Procedia Technology*, 24:980–987, 2016. International Conference on Emerging Trends in Engineering, Science and Technology (ICETEST - 2015).

[2] Michael Isard and Andrew Blake. Condensation – conditional density propagation for visual tracking. *INTERNATIONAL JOURNAL OF COMPUTER VISION*, 29:5–28, 1998.

[3] Zia Khan, T. Balch, and F. Dellaert. Efficient particle filter-based tracking of multiple interacting targets using an mrf-based motion model. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 1, pages 254–259 vol.1, 2003.