Tracking Blobs in the Turbulent Edge Plasma of Tokamak Fusion Reactors

Woonghee Han
MIT
Physics
harryhan@mit.edu

Randall A. Pietersen
MIT
CEE
rpieters@mit.edu

Rafael Villamor-Lora
MIT
CEE
villamor@mit.edu

Matthew Beveridge
MIT
EECS
mattbev@mit.edu

Nicola Offeddu
EPFL
Physics
nicola.offeddu@epfl.ch

Theodore Golfinopoulos
MIT
PSFC
golfit@mit.edu

Christian Theiler
EPFL
SPC
christian.theiler@epfl.ch

James L. Terry
MIT
PSFC
terry@psfc.mit.edu

Earl S. Marmar
MIT
PSFC
marmar@psfc.mit.edu

Iddo Drori
MIT
EECS
idrori@mit.edu

Abstract

The analysis of turbulent flows is a significant area in fusion plasma physics. Current theoretical models quantify the degree of turbulence based on the evolution of certain plasma density structures, called blobs. In this work we track the shape and the position of these blobs in high-frequency video data obtained from Gas Puff Imaging (GPI) diagnostics, by training a mask R-CNN model on synthetic data and testing on both synthetic and real data. As a result, our model effectively tracks blob structures on both synthetic and real experimental GPI data, showing its prospect as a powerful tool to estimate blob statistics linked with edge turbulence of the tokamak plasma.

1. Introduction

In tokamak fusion reactors, plasmas are magnetically confined and heated up to produce energy from nuclear fusion. In order to maximize the rate of fusion, it is vital to maintain an energy confinement as high as possible. The quality of this confinement is closely related to the turbulence at the edge region of the plasma core. Current theoretical models quantify the degree of this turbulence from the evolution of certain structures, known as blobs, within the plasma density field. This is an evolving area of research, and the different models require the analysis of different blob statistics that can be derived from image data (e.g. blob velocity, size, and intermittent). For example, the fluctuations in the plasma may be described by a stochastic model as the superposition of uncorrelated Lorentzian pulses, which is parameterized by the intermittency of blobs [4, 5]. Furthermore, the radial velocity and the size of blobs may be used to determine the theoretical blob propagation regime and the dependence of its properties on various plasma parameters [8]. Thus, it is crucial to have a reliable method for estimating statistics of blobs in order to study turbulences in the edge of tokamak plasmas.

In this work, we develop a deep neural network trained on synthetic data to track the shape and the position of blobs in experimentally obtained low-resolution (12 × 10 pixel), high-frequency (2 MHz) video data from Gas Puff Imaging (GPI) diagnostics [6, 12] on the Tokamak à Configuration Variable (TCV), shown on Figure 1. GPI is an edge diagnostic tool that measures the spatially-resolved fluctuations of plasma light emission, which can be used as a proxy for plasma density fluctuation measurements. Figure 2(b) shows a snapshot of the GPI data that captures a blob moving radially outwards after crossing the plasma edge (i.e. the Last Closed Flux Surface (LCFS)).

Currently, blob-like structures in GPI data are tracked by conventional algorithms such as brightness thresholding [9]. However, these methods do not generalize well due to the presence of measurement noise and varying experimental conditions. For instance, threshold values change between individual experiments, and experimental noise may result in structures that resemble blobs appearing in isolated frames. Conventional tracking algorithms fail to distinguish such noise fluctuations from actual blobs.
We therefore propose the use of a convolutional neural network (CNN) to track blobs, improving upon existing methods. While using CNN may be straightforward, our application domain introduces a level of complexity which may create a significant barrier to entry for non-domain experts. The current process of finding an appropriate algorithm, altering it for a specific use case, debugging errors, and then optimizing its performance to achieve useful results, is by no means trivial and may inhibit, or completely discourage, its use in certain contexts. Indeed, the associated time required by subject matter experts with limited coding experience, but who have a use for a specialized algorithm, is oftentimes very discouraging under the current paradigm. OpenAI Codex [2] is a Transformer trained on both text and code. Although it was originally intended as a means of converting textual descriptions, by designers for example, into code; in this work we apply it in a novel workflow so that non-domain experts may efficiently construct state-of-the-art models for our highly specialized domain.

Specifically, we use Codex to aid implementing a Mask Region-based Convolutional Neural Network (Mask R-CNN) [7] combined with a loss function that computes the pairwise cost between each object from the previous frame and each object in the current frame [3]. Our modified algorithm is able to segment blobs in a three-dimensional volume by preserving the spatiotemporal features in the GPI video data. We discuss the qualitative utility of using Codex to help with our code implementation, as well as present the results of this process with the model’s performance on both synthetic training data and real-experimental data.

Our methodology, including the model architecture, training approach, and hyperparameter tuning with Bayesian optimization, is described in Section 2. We then discuss the result of training and performance on experimental GPI data in Section 3, with our main conclusions summarized in Section 4.
2. Methodology

2.1. Base model: mask R-CNN

We implement a Mask R-CNN [7, 10] as a base model. The output from the Mask R-CNN consist of bounding boxes, masks, labels of the object class, and confidence scores. The model identifies three distinct classes: the background, the shear layer region, and blobs. Mask R-CNN is an extension of Faster Region-based CNN (Faster R-CNN) [11], which is designed to produce two outputs for each candidate object: a class label and a bounding box. Mask R-CNN extends this by an additional branch that predicts segmentation masks on each bounding box. The training is performed on a pre-trained off-the-shelf model.

2.2. Datasets and Training

Each experimental GPI dataset is composed of a series of 2D images, $12 \times 10 \times t$, where $t$ is the number of frames in the video. A snapshot of GPI data is shown in Figure 2(b) which shows a blob (as a bright spot on the bottom-right), and the LCFS, which is approximately the position of a shear layer, across which the plasma flow reverses direction. The masks of these two types of objects, blobs and shear layer, are what we aim to predict. Due to the limited amount of experimental data available and the current limitations for adequately labeling real world data, our model is trained using synthetic data. A snapshot of this synthetic data is shown in Figure 2(c). For these synthetic data, we are able to simulate complex instances which appear in the real GPI data, such as merging and splitting of blobs.

Although the GPI sensors output data with $12 \times 10$ pixels per frame, it is standard practice to use interpolation algorithms to upsample this resolution, in the present case to $256 \times 256$ pixels per frame. This degree of upsampling allows adequate detail during analysis without being too computationally demanding. Aligning with this convention, the synthetic data generated for training is also kept at $256 \times 256$ pixels per frame and the model is designed to receive inputs and produce outputs at this resolution.

2.3. Tracking-by-detection algorithm

After blobs are detected in each frame, temporal coherence between frames is enforced based on a tracking-by-detection workflow [3]. The algorithm of a typical tracking-by-detection method is illustrated in Figure 3, and consists of four steps:

1. **Object detection from the trained mask R-CNN model.** Given an input data, the bounding boxes and the masks of blobs are predicted for each frame by the trained mask R-CNN model.

2. **Feature extraction.** We select the objects to be tracked (i.e., blob objects). Predictions with scores below a certain threshold are discarded.

3. **Pairwise cost.** In this critical step, we exploit the temporal coherence of the images. Using a cost function, we compute the pairwise cost between the objects in the current and previous frames. There are different alternatives for the cost function, and we use the volumetric intersection over union (VIOU) as described in Section 2.4.

4. **Matching.** Using the cost matrix from the previous step, we assign unique correspondence between objects with the constraint that no object receives more than one assignment. If a new object appears in an isolated frame (i.e. it has no correspondence either in the previous or the next frame), then the object is ignored and discarded as a noise fluctuation. Moreover, when a new object appears in the current frame with no correspondence in the previous frame, but a match in the next frame, we start a new tracklet [3]. By bookkeeping of the active and finished tracklets we are able to count the number of blobs in the video and record their trajectories.

2.4. Volumetric IoU

To quantify the loss during (1) the training of the Mask R-CNN detector, and (2) the pairwise cost step in the tracking-by-detection algorithm, we use volumetric IoU as our evaluation metric. In object detection algorithms it is customary to use intersection over union (IoU) metrics to evaluate the goodness of the prediction. For the prediction/ground-truth pair (e.g., bounding boxes, mask), the IoU is computed as the ratio between the area of intersection and the area of the union. While this approach is quite robust when dealing with the mask detection of solid objects with well-defined, sharp boundaries, IoU can mislead the score for our application when the area of intersection has low brightness level which is intuitively a poor prediction.

Note that the boundary definition of blob structures within the plasma is a function of the brightness threshold. Since this threshold may vary from experiment to experiment, in our case, it is more appropriate to describe the blob shape as a volume, with volumetric IoU (VIOU), being the brightness level the third axis, as shown in Figure 4.

2.5. Data Augmentation

In order to improve the generalization performance of the model, a set of data augmentation transformations are applied to the training data with corresponding probabilities as shown in Figure 5. Specifically, given random numbers $x_{\text{horizontalFlip}}$, $x_{\text{scale}}$, $x_{\text{translate}}$, $x_{\text{shear}}$, and $x_{\text{rotate}}$ between 0.0 and 1.0, we perform the following transformations:
Figure 3. Tracking-by-detection algorithm consisting of four steps: (1) Object detection within each frame using a pre-trained detector. Here the mask R-CNN produces three outputs for each candidate object: a label, a bounding box, and a mask. The model distinguishes between three classes: background, shear layer, and blobs. (2) Extraction of features of interest (i.e. blob objects). (3) Computation of pairwise costs between objects in the current and previous frame. (4) Matching between object assigning unique correspondence.

Figure 4. Volumetric Intersection over Union (VIoU) computed for a prediction on an image containing a blob (top) using the volume under the profile of the brightness (bottom).

- Flip: horizontally flips with \( f_{\text{horizontalFlip}} \) probability when \( x_{\text{horizontalFlip}} < P_{\text{horizontalFlip}} \).
- Scale: scales with random scaling factor between 0.8 and 1.2 when \( x_{\text{scale}} < P_{\text{scale}} \).
- Translation: translates with random translating factor between 0.0 and 0.2 when \( x_{\text{translate}} < P_{\text{translate}} \).
- Shear: shears with random shearing factor between \(-0.2\) and \(+0.2\) when \( x_{\text{shear}} < P_{\text{shear}} \), and
- Rotation: rotates with random angle between \(-10\) and \(+10\) deg when \( x_{\text{rotate}} < P_{\text{rotate}} \).

The transformation probabilities \( P_{\text{horizontalFlip}}, P_{\text{scale}}, P_{\text{translate}}, P_{\text{shear}}, \) and \( P_{\text{rotate}} \) are hyperparameters which are optimized by Bayesian optimization as described in Section 2.6.
2.6. Bayesian Optimization of Hyperparameters

Hyperparameter tuning is a key step in optimizing the performance of machine learning algorithms, however, the computational demands of such an analysis vary greatly depending on the dimensionality of the parameter space and complexity of the algorithm being evaluated. In our model, Bayesian optimization is used to find the hyperparameter values that minimize the loss of the CNN trained to detect blob objects in GPI data.

We apply Bayesian optimization [1] as two levels of hyperparameters: (1) learning rate, momentum, weight decay, and number of epochs; and (2) data augmentation transformations $P_{\text{horizontalFlip}}$, $P_{\text{scale}}$, $P_{\text{translate}}$, $P_{\text{shear}}$, $P_{\text{rotate}}$, as well as dropout probability $P_{\text{dropout}}$ for each dropout layers added after ReLU layers in later parts of the model which detect regions of interest.

The exploration ranges for each hyperparameters are shown in Table 1. We first optimize first the group one hyperparameters, and then keep those values fixed while optimizing the second group of hyperparameters.

2.7. Co-programming using Codex

OpenAI Codex [2] is a large scale Transformer trained on both text and code. Recently, the toolbox of modern coding has been extended to include using OpenAI Codex or Github co-Pilot to aid and guide the implementation process. In the spirit of best coding practices and these advances in automation, this work is the first to use program synthesis to improve performance by automatically writing architectural components and hyperparameters. To do so we provide prompts including part of our model code to OpenAI’s Codex [2] and incorporate the architectural and hyperparameter changes generated. Therefore, instead of setting the model architecture and hyperparameter values on an ad hoc basis, we use OpenAI Codex. For ad hoc values, the horizontal flipping factor $f_{\text{horizontalFlip}}$ (probability of flipping when the horizontal flip is practiced) and scaling factor (percentage of the scaling) in data augmentation were suggested by Codex as 1.0 and 0.5, respectively. For model structures, Codex suggests replacing the ReLu layers with leaky ReLu layers, and plain dropout layers (randomly zeroes some of the elements of the input tensor) with 2D dropout layers (randomly zeroes entire 2D channels of the input tensor, e.g., 2D with the channel and the batch).

We find that providing guidance through human code snippets allows Codex to make these decisions by automatically modifying existing code. These decisions would have been challenging for programmers to make unless they try costly runs for every different combinations, without the aid of Codex.

2.8. Conventional Algorithm for Labeling Experimental GPI Data

One of the aims of this study is to improve upon the performance of the conventional tracking algorithm based on image thresholding [9]. The conventional tracking algorithm generates blob sizes and trajectories from GPI data, by tracking the contours that exceed a user-defined threshold. With these, one can directly determine the distributions of speed, size, and frequency of blobs. While this algorithm is a common approach for blob analysis, it inherently has two primary shortcomings that our approach is designed to address: (i) It does not generalize well across multiple experiments and variations in data, which requires to carefully oversee the tracking process and continuously adjust values during data processing steps. Implementing a
Mask R-CNN model for our approach makes this process more invariant to changes in blob representation. (ii) It is incapable of instance-level object recognition and cannot adequately characterize the merging and splitting of blobs, which is critical for the understanding of the nature of the plasma edge turbulence. Mask R-CNN is able to perform this instance-level recognition [3]. Although this conventional algorithm is what our model aims to outperform, we used it for labeling blobs in experimental GPI data in order to test the performance of our model (trained with synthetic data) on the real GPI data.

**Uncertainty estimates.** We estimate uncertainty using dropout by training the network using dropout and then testing each example by running multiple forward passes with dropout weights.

3. Result

3.1. Result of Bayesian optimization

As described in Section 2.6, hyperparameters are tuned by two levels of Bayesian optimization (BO) with each of the exploration ranges shown in Table 1. The optimal values and importance found for each hyperparameter are summarized in Table 2 and 3. The convergence of VIoU in the hyperparameter space is shown for the three most important hyperparameters in BO level 1 (Figure 6 (a)) and in BO level 2 (Figure 6 (b)). Here, the VIoU score driving BO is $0.7 \times \text{VIoU}_{\text{blob}} + 0.3 \times \text{VIoU}_{\text{shear}}$, where $\text{VIoU}_{\text{blob}}$ and $\text{VIoU}_{\text{shear}}$ are the average volumetric IoU for blob and shear layer predictions, respectively. This weighting emphasizes the importance of blob predictions over predictions for the shear layer. The importance is estimated as the mutual information regression for the resulting VIoU score from each BO trial. For BO level 1, the learning rate reduction period is the most important hyperparameter. The VIoU increases from 0.77 and converges to 0.89 as shown in Figure 6 (a). For BO level 2, the optima in Table 3 are shallow since there is not much change in VIoU for trials shown in Figure 6 (b). The VIoU’s are around 0.89, which is similar to the convergent value of BO level 1. Our conclusion from these results is that we found optimal values of hyperparameters in BO level 1 which greatly improve the performance of the model. In contrast, we conclude that data augmentation, as well as the addition of dropout layers to the model, have not much effect on the overall performance of the model.

3.2. Training of the model on synthetic data with optimized hyperparameters

The model is trained on synthetic data using the optimized hyperparameters found in Section 3.1, and the result is shown in Figure 7. The size of the training dataset is twice as large as the dataset used in Bayesian optimization. Figure 7 shows VIoU’s evaluated with held-out test data (20% of the dataset) for each epoch for blob and shear layer predictions. The uncertainties are estimated by Monte Carlo (MC) dropout with dropout probability 0.3 and 100 MC passes. The score gradually increases up to $\sim 0.88$ for blob predictions and $\sim 0.80$ for blob predictions. The uncertainty also gets smaller in the later epochs (0.08 for blob predictions).
Momentum
0.2
0.4
0.6
0.8
1.0 Weight decay
0.0000
0.0002
0.0004
0.0006
0.0008
reduction period
10
20
30
Learning rate
Ptranslate
0.0
0.2
0.4
0.6
0.8
PhorizontalFlip
0.2
0.4
0.6
0.8
Pdropout
0.0
0.1
0.2
0.3
0.4
Volumetric IoU of BO trial
Level 1
Bayesian Optimization
Level 2
Bayesian Optimization
Figure 6. Convergence of volumetric IoU score in (a) BO level 1 and (b) BO level 2. In both cases, the three most important hyperparameters are plotted.

Figure 7. Result of the training of the model showing volumetric IoU’s from test data for each epoch, for blob and shear layer predictions.

and 0.04 for shear layer) compared to the earlier. As a conclusion, the model trained on synthetic data with the optimized hyperparameters does achieve a high score in testing with synthetic data, comparable to the score achieved in the Bayesian optimization.

3.3. Testing of the trained model on real experimental GPI data

Using the resultant weight matrices for the model trained on synthetic data, real experimental GPI data is then inputted to test performance. While performance metrics for the experimental data are valuable, it is important to understand that unlike the synthetic data, the conventional tracking algorithm used to generate ground truth labels for the experimental data often has errors and may therefore be unreliable. One such example is shown in Figure 8, where the black mask, which is the label from the conventional algorithm, is inaccurate compared with the red mask, which is the prediction from the model. In this case, the VIoU between black and red masks is low but the ground truth is not good enough, which makes the model’s performance to be underestimated from this low VIoU. For this reason, only VIoU’s higher than 0.5 are considered. The mean of such VIoU’s is 0.61 with a standard deviation of 0.08, so the model performs fairly well on experimental GPI data. Figure 9 shows the model’s tracking ability for the case of splitting blobs, which is not captured by the conventional tracking algorithm. Here, the blob ID in pre-splitting (Blob 16) is preserved after the splitting and a split-off blob has a new ID (Blob 17), showing the efficacy of the tracking-by-detection algorithm described in Section 2.3. In conclusion, these results show good performance of the model on real experimental GPI data, and these also highlight the limitations of the conventional tracking algorithm which is overcome by our proposed model.

4. Conclusion

We present a model for blob tracking which is trained with hyperparameters found by Bayesian Optimization and an architecture design guided by OpenAI Codex for improving performance. It effectively predicts the shape and the location of turbulent structures in the edge of tokamak fusion reactors, called blobs, in both synthetic and real experimental GPI data. This new model will serve as a reliable tool for tracking blobs, which allows estimation of various statistics of blob dynamics connected to the level of turbulence in the edge of tokamak plasmas. Having this characterization of the edge turbulence based on blob statistics helps us understand turbulent modes which are harmful to the plasma confinement, thereby improving the generation of fusion energy.
Figure 8. An example of experimental GPI data with a ground truth mask (black) labeled by conventional tracking algorithm which is not as good as predicted mask (red) by the model.

References

[1] Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. Botorch: A framework for efficient monte-carlo bayesian optimization, 2020. 5
[2] Mark Chen et al. Evaluating large language models trained on code. arXiv preprint 2107.03374, 2021. 2, 5
[3] Ben Cook. Object tracking in 75 lines of code, Aug 2020. 2, 3, 6
[4] O. E. Garcia, R. Kube, A. Theodorsen, and H. L. Pécseli. Stochastic modelling of intermittent fluctuations in the scrape-off layer: Correlations, distributions, level crossings, and moment estimation. Physics of Plasmas, 23(5):052308, 2016. 1
[5] O. E. Garcia and A. Theodorsen. Power law spectra and intermittent fluctuations due to uncorrelated lorentzian pulses. Physics of Plasmas, 24(2):020704, 2017. 1
[6] W. Han, N. Offeddu, T. Golfinopoulos, C. Theiler, C. K. Tsui, J.A. Boedo, E.S. Marmar, and the TCV Team. Suppression of first-wall interaction in negative triangularity plasmas on TCV. Nuclear Fusion, 61(3):034003, feb 2021. 1
[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In Proceedings of the IEEE international conference on computer vision, pages 2961–2969, 2017. 2, 3
[8] J. R. Myra, D. A. D’Ippolito, D. P. Stotler, S. J. Zweben, B. P. LeBlanc, J. E. Menard, R. J. Maqueda, and J. Boedo. Blob birth and transport in the tokamak edge plasma: Analysis of imaging data. Physics of Plasmas, 13(9):092509, 2006. 1
[9] N. Offeddu et al. Cross-field and parallel dynamics of SOL filaments in TCV. In preparation. 1, 5
[10] PyTorch. Torchvision object detection finetuning tutorial, 2020. 3
[11] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. arXiv preprint arXiv:1506.01497, 2015. 3
[12] S. J. Zweben, J. L. Terry, D. P. Stotler, and R. J. Maqueda. Invited review article: Gas puff imaging diagnostics of edge plasma turbulence in magnetic fusion devices. Review of Scientific Instruments, 88(4):041101, 2017. 1

Figure 9. The model (red) captures the moment of a blob before splitting (upper) and after splitting (lower) while the conventional tracking algorithm (black) does not.