Orbital Debris Classification Using Unsupervised Learning for Space Situational Awareness

Gregor Limstrom
limstrom@umich.edu

Gabriel Ronan
gronan@umich.edu

Justin Wang
justwang@umich.edu

Abstract

Human space exploration has left significant amounts of uncontrolled objects in space throughout the last half-century, increasing the risk of current and future space operations. This debris is traditionally tracked through ground based radar telescope stations, but as the amount of small debris produced by collisions increases, it is critical to track and identify debris using onboard orbital sensors, as small debris can't be detected through conventional ground based telescopes, but still poses a very serious threat to current and future orbital operations. We propose to identify debris through a lightweight computer vision method by extracting features in time-fused camera data and highlighting debris objects using K-Means and Gaussian Mixture Modeling. **Project code can be found at** `https://github.com/graygr/eecs-545-project`

1 Introduction

Orbital debris is a significant problem that is predicted to worsen in the near future from ambitious constellation launch plans. The debris in question is defined as any man-made object in orbit around the earth that no longer serves a useful function, and most notably includes decommissioned spacecraft, abandoned launch vehicle stages, mission-related debris, and fragments of these assets. Currently, there are an estimated 36,500 pieces of debris in orbit that are larger than 10 cm, and, though smaller pieces can compromise an active satellite, they are less hazardous and not detectable with current ground sensing capabilities. [1]

Orbital debris not only damages satellites and other objects of interest; it also creates more debris in a cascading effect that can increase the amount of debris exponentially. This idea was proposed by NASA scientist Donald J. Kessler in 1978 and is called Kessler Syndrome, which describes an effective chain reaction where debris collisions can cause uncontrollable fragmentation. In 2009, Kessler wrote that modeling results indicate that the Earth orbit debris environment was already unstable, "such that any attempt to achieve a growth-free debris environment by eliminating sources of past debris will likely fail because fragments from future collisions will be generated faster than atmospheric drag will remove them". [7] Active satellites are of high value and importance to different nations around the world for communications, GPS, and imaging purposes. Recent news points to orbital debris being an increasing threat to current and future space missions. On January 18th, 2022, the Space Debris Monitoring and Application Center of the China National Space Administration (CNSA) issued a warning of a dangerous encounter between the Tsinghua Science Satellite and one of more than a thousand trackable pieces of debris from the Nov. 15 Russian ASAT (Anti-Satellite weapons) test. [3] The Chinese satellite encounter is just one of the many recent spacecraft near-encounters with space debris.

Many of these pieces of debris remain a threat in orbit for hundreds or even thousands of years. While this will not render future space exploration impossible, it will drastically increase the cost and hazard of space exploration. Currently, If we solve this problem and increase the speed and accuracy of the identification methods, we can help contribute towards solving the first big hurdle of removing space debris, and beginning a new era of responsible orbital management. This technology can also be brought towards current active satellites, allowing them to potentially identify dangerous small debris and maneuver to avoid them, extending their mission lifetime. In addition, this technology is needed in order to pursue active debris removal in order to detect and remove debris.

2 Related Work

Historically, most debris has been tracked using ground based radar and optical telescopes, with exhaustive detection algorithms running on high performance computers. These algorithms work by combing through radar and optical imagery in order to find incongruencies, indicating debris. However, as more intra-debris collisions continue to occur, more small debris will make up the orbital environment, and conventional telescopes can only detect around a 10cm sphere in Low Earth Orbit (LEO), or around a 1m sphere in Geostationary Earth Orbit (GEO). This small debris still poses a serious threat to current and future space operations, as even a small 1cm object will puncture most orbital shielding.

In [7], Zamani et. al explore a novel vision based algorithm for debris detection using onboard optical sensors. The dataset is generated from a closed source simulator at NASA Marshall Space Flight Center, but the video output is provided. Between each frame, centroids of each debris are extracted, then compared to the next frame to compute the most likely translation for each cloud of debris. From these translation vectors, the most likely outliers are selected and classified as debris.

In our approach, we chose to attempt to improve on the approach taken in [7] by taking a computer vision based approach. Rather than identifying translation vectors in a frame-by-frame manner, we will extract cheap, latent features from fusing time series data together in order to capture the movement of the objects through time. This approach will allow us to be more flexible with our compute load with variable frame stride, and more robust detection in complicated and chaotic scenarios due to the additional features and unsupervised learning techniques we implement.

At first, we attempted to use object detection models to identify debris. By first locating the fused blobs, we planned to draw bounding boxes around them and classify these individual patches. We studied several object detection models; first of all, in the paper [2], Girshick et al. proposed the revolutionary R-CNN. It uses "Selective Search" to find around 2000 region proposals that likely contain an object, which significantly lowers the cost of sliding windows. Each proposal is resized into same-size rectangles and fed into CNN models such as VGG-16[6] for feature extraction. With these features, a Support Vector Machine can classify them into given categories. On the other hand, linear regression is applied to the proposals to maximize their intersection over union with ground truth bounding boxes. However, we were seeking a real-time solution for debris classification, so R-CNN was not adopted.

We did further research into the R-CNN, and then we found the Faster R-CNN [5], which uses Region Proposal Network (RPN) and Anchor Boxes to predict bounding boxes instead of the time-consuming selective search. It is significantly faster than R-CNN, but it is still not capable of real-time classification. Thus, we decided to use YOLO [4], a model that can perform in almost real-time. Compared to Faster R-CNN, YOLO shares the same neural network for RPN and classification, making it significantly more efficient. However, we faced two issues when trying to apply YOLO to our task. First, the data we are using does not have any label. This is a huge problem since all object detection models are supervised learning. Second, debris has a different pattern for different data. For example, in the [7], debris moves faster than stars in the *Simple Background* video, while stars move faster than debris in *Simple Foreground*. In this case, if an object detection model is trained with one of the data, it will not be able to generalize to another one. We decided to pursue an unsupervised learning method due to the different situations presented and the large amount of unclassified data.

3 Data

Finding data for this problem was difficult, as many current efforts for researching this are being conducted by government organizations. Most, if not all satellite data is under lock and key as well.

Thankfully, [7] provides their video data set from a state of the art simulator from NASA Marshall. The simulator is closed source from our research, so we worked with the provided video files.

Table 1: Input Sample Data Specification

Sample Data	Frame Rate (fps)	Duration (sec)	Frame Count	Spots (mean ±std. dev)	Debris (mean ±std. dev)
Simple Background	60	60	3600	118.3 ±2.1	5.3 ±2.1
Simple Foreground	30	30	180	134.7 ±2.4	1.3 ±0.7
Complex Background	60	60	3600	114 ±4.8	5.1 ±2.3
Complex Foreground	30	30	900	117.7 ±3.7	6.8 ±2.0

Each video is 1920x1080 pixels, stored in Grayscale. 1 details more specifics. Each video tests a different aspect of the classifier, and will allow us to test which features are most effective at classifying outliers.

Simple Background contains static stars with debris moving over them. *Complex Background* contains moving stars with the same debris pattern. *Simple Foreground* contains a small amount of debris with a moving background. *Complex Foreground* contains a higher number of debris moving at various different velocities and directions with the same background.

4 Learning Pipeline

In this section we will cover the components and functionality of our learning pipeline. As described below in 1, each camera frame is processed, fused with prior times, then contours and features are extracted and classified.

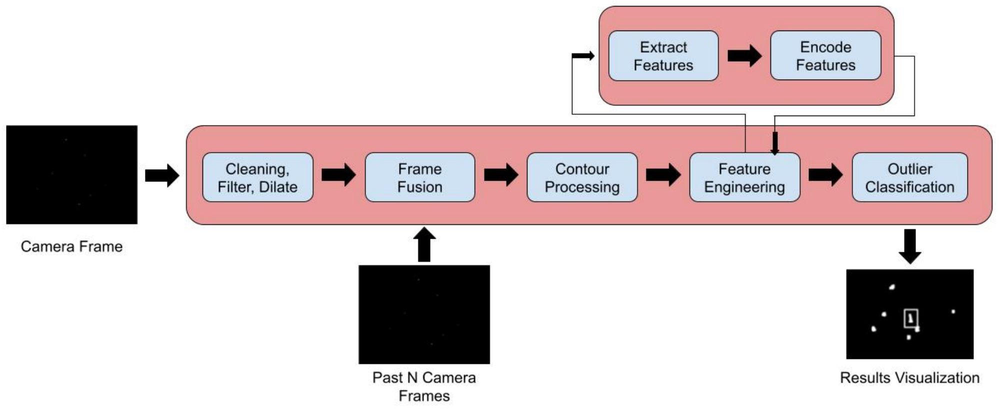

Figure 1: Data Pipeline

4.1 Data Pre-Processing

In order to increase the clarity of our data features, we needed to perform some pre-processing on the image. As mentioned above, each frame of the video contains 1920x1080 pixels, and each object is represented in this matrix as a cluster of some level of light. This brightness level is determined by the estimated emitted photon flux of the star or debris, but in order to make feature extraction possible, we must process it. The hyperparameters in this step include the binarization threshold, the size and shape of the kernel, and the number of frames we are fusing for each classification, as well as the frame stride.

First, we run a binary filter with a threshold of 50, which simply sets each pixel to 0 if it's value is below the threshold, and 255 if it is greater than or equal to the threshold. This removes most background noise and insignificant sources of light from the images, and emphasizes the debris we are targeting.

Second, we use a square kernel to dilate the image. We use a kernel with a diameter of 5 pixels, which allows the classifier to catch most fast moving objects and classify them properly as a single contour. The dilation kernel simply sets all pixels within the area to 255 if any pixel within them is equal to 255.

Last, in order to capture the movement of the debris through time, we fuse the past N frames together in order to capture each distinct object's trace through space. This is accomplished through a matrix sum followed by a normalization in order to keep pixel values within their acceptable limits. The frames are read and stored in a circular array in order to have continuous access to prior frames within our memory window. We can also alter the frame stride in order to reduce the number of times we consider a frame of data in order to reduce computational time without necessarily sacrificing classification accuracy.

4.2 Feature Extraction

At this point in the pipeline, we are working with an image composed of many white blobs on a black background, and need to extract information about each blob in order to classify it. We considered several different options, but chose to use OpenCV's contour extraction function, as it was very fast, but allowed us more direct control of the features we extract from the contours for our classification.

A contour is a group of white pixels separated from other pixels. For our purposes, each potential debris object is a contour on the black and white display. In order to classify these contours into debris or not, we needed to extract certain characteristics to help disambiguate them.

The first characteristic we chose is the pixel area, which is a simple count of the total number of pixels in the contour. This should capture the motion of the object through the field of view, as the faster an object is moving, the larger area its trace will have.

The second characteristic we chose is the perimeter of the object, which is calculated by counting each edge pixel of the contour. This should help us differentiate between objects with large movement paths and stationary objects with a larger photon flux, such as a star. The objects with large movement paths will have a relatively small area compared to their perimeter, whereas the objects with a larger light output but less movement will have a larger area relative to their perimeter.

The third characteristic we chose is the orientation angle of the object, which is the rotation angle of the best-fit ellipse relative to the zero-line. We calculate this by taking the two extreme (x,y) locations of the image and finding the slope between them. This calculation will allow us to estimate the direction the object is travelling in, which will help disambiguate debris when it is travelling in a direction normal to the movement of the background star field.

The fourth characteristic is the aspect ratio of the bounding box around the contour, which is simply its width divided by its height. We believe this will help us capture the arc of movement better, which is observed when the stars are rotating overhead.

The final characteristic we chose is the radius of the minimum enclosing circle around each contour. This is calculated by finding the two farthest pixels from each other in each contour, then using half that distance as the radius. We believe this will help capture the velocity aspect of the object and bolster the importance of that measurement with a similar but distinct feature to perimeter.

4.3 Classification

Once we have extracted our features, we used several unsupervised learning methods to classify the outliers. Since the star field is primarily made up of stars, most data points will have similar behavior due to their extreme distance from the observer. Debris will naturally lie on the outer edges of the feature space. We used several different classification methods in order to test classification accuracy and computational load.

4.3.1 N-Dimensional Distance With L2 Norm Kernel (Naive Means Method)

The first method of classification we chose was to discriminate based on the distance in N-Dimensional feature space using the L2 Norm to calculate the error from the mean feature vector for the cumulative frame. We will refer to this as the Naive Means method from now on.

The cost function is hence the L2 norm from the mean. We calculate the mean by averaging the features of each contour in the combined frame, and then computing a weighted mean with the prior mean multiplied by a decay constant. This allows for more stability in run time, but provides mechanisms for the system to self correct if the background changes suddenly, such as during a flight maneuver.

Our intuition with this method is that it may not be the most effective way of classifying the debris, as each error is weighted identically, but it will prove to be a demonstrative baseline in order to quantify how successful our features are at discriminating the contours present in the data.

4.3.2 K-Means

K-Means is an iterative clustering technique that partitions n observations into k clusters. Each observation belongs to the cluster with the nearest mean (cluster center/centroid). Distances to each cluster center are calculated using Euclidean distance. Due to this distance metric, the optimal cluster shape is a sphere of dimension equal to the dimension of each observation, or the number of feature vectors for each observation. To train the K-Means clustering algorithm, we used two methods. The first was to use all five features: pixel area, perimeter, orientation angle, radius of the minimum enclosing circle, and aspect ratio of the enclosing ellipse. The second was to use the distance kernel function to reduce all features down to scalar values, representing the euclidean distance of the feature vector to the origin. This second method was proposed in hopes of identifying slower moving debris objects.

4.3.3 Gaussian Mixture Model

Gaussian Mixture Model (GMM) is another clustering technique besides K-Means. GMM is based on probability density estimation and Expectation-Maximization (EM). Instead of just finding a single point in the feature space as a cluster center and assigning points to the closest center based on the euclidean distance, GMM clusters data points as they are Gaussians. This method has several advantages over K-Means. The first advantage is that it can better classify clusters with non-spherical shapes since it's not using euclidean distance from a center point. The second advantage is that GMM can separate different clusters based on their distribution when they have an overlapping center. Moreover, GMM is a generative model, i.e., it returns a probability model, which can be used to compare sets of data to see if they differ and can be very useful for our project.

4.4 Cross Validation

To evaluate the generalization ability of our model, we use K-fold Cross-Validation method on both K-means and Gaussian Mixture Model where $K = 10$. We perform K-fold by first pickling all the features from each video, and then shuffle the features to ensure the independence between each fold. With the shuffled features, we divide them into K equal segments. In Each fold, one of the segment will be separated from the shuffled array to be used as a validation set, The model we are evaluating will fit the remaining features and perform classification on the validation segment. Since there are no overlapping between each fold, we sum up the classification result unshuffled, i.e., sum them up in an chronological order. After K folds, we can obtain the classification result of all the data point.

5 Experimental Results

The general feature hyperparameters we used are described in Table **??**. These were derived through trial and error in order to find the set of features that most consistently separated the data set.

Table 2: Feature Hyperparameters

Num. Frames	Frame Stride	Kernel Size
10	3	5

5.1 Runtime Analysis

Runtime analysis was conducted on a single computer for consistency. Output was disabled in order to maximize performance. The results can be seen in Table 3 below.

Table 3: Simple Background Performance Statistics

Algorithm / Dataset	Runtime Duration (sec)	Frame Process Rate (sec/frame) $\pm std.dev$
Simple Background		
AMOS 2019	27.85	0.0077 $\pm$0.001
N-D L2	315.4	0.088 $\pm$0.1
K-Means	523.8	0.145 $\pm$0.01
GMM	176.4	0.049 $\pm$0.01
Simple Foreground		
AMOS 2019	1.41	0.0078 $\pm$0.001
N-D L2	16.18	0.089 $\pm$0.01
K-Means	28.4	0.158 $\pm$0.01
GMM	9.72	0.054 $\pm$0.01
Complex Background		
AMOS 2019	26.5	0.0073 $\pm$0.001
N-D L2	313.1	0.087 $\pm$0.01
K-Means	538.1	0.149 $\pm$0.01
GMM	187.2	0.052 $\pm$0.01
Complex Foreground		
AMOS 2019	6.13	0.0068 $\pm$0.001
N-D L2	111.67	0.12 $\pm$0.02
K-Means	130.4	0.145 $\pm$0.01
GMM	58.53	0.061 $\pm$0.01

5.2 Classifier Accuracy

Classifier Accuracy was compared against the results of the classifier in [7]. Ground truth data was sampled at a rate of 1hz and then interpolated.

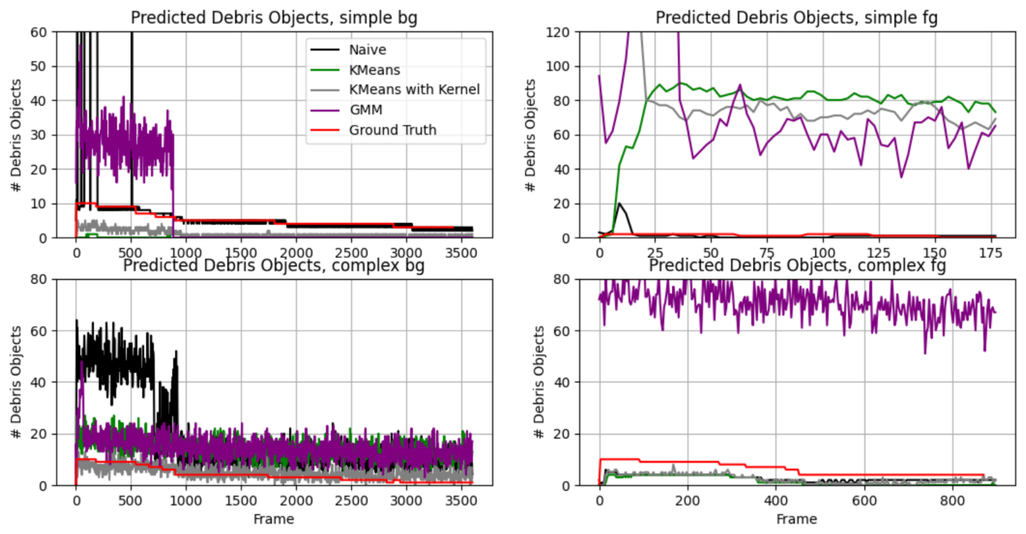

Figure 2: Classifier Performance vs. Groundtruth

The Naive Means method had one hyperparameter, error distance threshold, which was the scalar multiple of how many standard deviations away from the mean classified as a debris. This was tuned manually, and varied between data sets. This allowed us to specifically define the variation cutoff for debris.

We tuned the number of clusters, the number of iterations, and the types of features used to train the K-Means algorithm. Unlike stars and other unknown objects, the debris objects were not always grouped into a single cluster, therefore it was not always possible to capture all debris objects using the features that we extracted using 2 clusters. However, When the number of clusters was set to 3, too many non-debris objects became classified as debris, and it was the 2 cluster parameter that minimized the error. The number of iterations was set to 10000 to allow the algorithm time to converge, but larger values than this would result in poorer real time performance. Applying a kernel to the feature vectors significantly improved the classification accuracy of the complex-bg data set, but failed to significantly improve classification for the other data sets. The accuracy of classifying simple-bg and simple-fg improved slightly, while complex-fg remained the same.

We tuned three hyperparameters of GMM, i.e., number of components, maximum iterations, and covariance type. Tuning the number of components yields similar result of the k-means model. Once we set it to 3, a large portion of stars were classified incorrectly. Thus, we kept the number of components to 2. For the maximum iterations, we set it to 10,000 since larger numbers will make the model unable to reach a real-time performance. Lastly, we experimented the covariance type in ’full’ and ’diag’. We did not notice significant differences between these two types of setting.

6 Discussion and Future Work

From our results, we believe our feature extraction was a mixed success. In some cases, such as the simple background, it was able to easily differentiate between debris and background objects. However, with more complex data, our feature space struggled to define a clear line between debris and stars. We believe that this is because the features we chose emphasize the outlying features of rapidly moving debris contrary to the movement of the star field, but by using these features, the debris that lies closer in feature space to the average star is much harder to differentiate.

6.1 Error analysis

In each method, our runtime performance was below par for the data parameters we chose, but we expected this outcome. The baseline runtime of the AMOS project is low because it is well optimized using parallel computing techniques and a much more efficient language. From our code profiling analysis, we spend a large amount of time in Python’s loops in order to calculate our features and means. Given more time, we wish to convert the project to C++ and properly optimize it, which we estimate will reduce our runtime to 1/100th of its current duration.

6.1.1 Naive Means Method

The Naive Means method was very successful in the case of the simple background, outperforming the others and almost perfectly matching the groundtruth, but became very sensitive and inconsistent with grouping with more complicated data sets. When tuning the sensitivity, we found that some debris often overlapped with stars with the distance metric we used, which was evident in our results on the complex datasets. However, it was easily outperformed by the more advanced methods below, both in timing and accuracy, as it struggled to maintain a boundary between debris and stars and oscillated and misclassified them. However, through manual tuning we were able to prevent it from underfitting the data and overpredicting debris.

6.1.2 Gaussian Mixture Model

After obtaining the K-Fold result of GMM, we found that it only has decent performance on the *Complex Background* data.The most reasonable explanation of this phenomenon is that the features we chose did not separate the data into the form of Gaussian distribution. Since the GMM assumes the data are consists of different Gaussian distribution, we have to find other feature that best describe the difference between stars and debris. Thus, finding a better feature becomes our main future goal.

6.2 Future Work

Unfortunately we were not able to explore every potential avenue with this project due to time constraints, as well as two of the five members of our team dropping the class as we began heavy work into the project. There are areas where we think we could improve the project and demonstrate the potential to perform in real world scenarios.

The first, and initially planned improvement was to use a Convolution Neural Network (CNN) in order to extract latent features within the data and use it to improve the classification performance. However, CNN's require labelled training data, and it proved too labor intensive to individually label each point frame by frame. The team also decided that a fixed training method for debris detection is likely not able to work in a dynamic orbital environment with no fixed reference point, so if the spacecraft changes its orientation, the previously trained model can prove to be useless. However, we believe that there are still potential CNN applications, they may just need to be trained for certain spacecraft orientations, and this proved to be beyond the scope of our project.

The second potential improvement is utilizing manual data modification in order to test the potential of our detection system. As mentioned previously in the paper, the simulator that our data is generated from is closed source and private, but we could still potentially add new pieces of debris in more challenging scenarios to see if our detector is able to find them. One such case as an object with a velocity vector parallel to the observer spacecraft. In this case, the object would appear to be still, or moving very little. This would be difficult to identify, and we could fuse simulated radar or lidar measurements in order to determine if it is debris.

Lastly, we would ideally run this algorithm on simulated hardware in order to test performance on a realistic compute power. This would likely require rewriting the code base in C++ in order to improve our performance, but in order for this to be usable in a real world situation, it must perform close to real time, or roughly 10-30 hz.

7 Conclusion and Broader Impact

While we may not have been successful in our attempt to improve on the work of [7], we learned many valuable lessons in real world machine learning implementations, such as data processing, the different sensitivities of classifiers, and run time optimization. Feature extraction is a difficult process that we did not execute perfectly, but we were able to achieve a level of success and experience an interesting challenge that taught us more about the real world issues that machine learning engineers must face.

We hope to apply the results of this project and the lessons we learned to the field of orbital debris removal through Gregor's research project with Northrop Grumman exploring the creation of a manual system of debris removal. We believe optical data is insufficient to reliably capture all debris, especially the debris that we will likely target and approach, thus an additional sensor such as radar or lidar is needed in order to add extra features to differentiate debris moving with a low relative velocity. Unsupervised learning provides a powerful tool to detect outliers, and we hope to see this approach applied to solve real world issues threatening the orbital environment of tomorrow.

References

[1] European space agency: Space environment statistics 04/2022. `https://sdup.esoc.esa.int/discosweb/statistics`, Apr. 2022.

[2] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[3] B. Li, J. Huang, Y. Feng, F. Wang, and J. Sang. A machine learning-based approach for improved orbit predictions of leo space debris with sparse tracking data from a single station. *IEEE Transactions on Aerospace and Electronic Systems*, 56(6):Article number: 90760324253–4268, December 2020.

[4] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[5] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[6] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[7] Y. Zamani, J. Amert, T. Bryan, and N. Nategh. A Robust Vision-based Algorithm for Detecting and Classifying Small Orbital Debris Using On-board Optical Cameras. In S. Ryan, editor, *Advanced Maui Optical and Space Surveillance Technologies Conference*, page 91, Sept. 2019.