Stone Soup: Announcement of Beta Release of an Open-Source Framework for Tracking and State Estimation

David Last[a], Paul Thomas[b*], Steven Hiscocks[b], Jordi Barr[b], David Kirkland[c], Mamoon Rashid[c], Sang Bin Li[c], and Lyudmil Vladimirov[d]

[a] *Air Force Research Laboratory (United States)*
[b] *Defence Science and Technology Laboratory (United Kingdom)*
[c] *Defence Research and Development Canada (Canada)*
[d] *University of Liverpool (United Kingdom)*
[*] *Corresponding Author - email: pathomas@dstl.gov.uk*

ABSTRACT

Tracking and state estimation technologies are used in a variety of domains that include astronomy, air surveillance, maritime situational awareness, biology, and the internet. Algorithms for tracking and state estimation are becoming increasingly complex and it is difficult for researchers and skilled practitioners to implement and systematically evaluate these state-of-the-art algorithms. System designers also need to objectively assess the performance of algorithms against operational requirements, and tools to conveniently perform such systematic assessment have been lacking. Recognising this problem, an initiative was taken to create an open-source framework called "Stone Soup", which would be used for the development, demonstration, and evaluation of tracking and state estimation algorithms. Stone Soup was made openly available in April 2019 as a beta version (V0.1b1). This paper introduces the Stone Soup framework and describes how users can take advantage of this framework to develop their own algorithms, set up experiments with real-world data, and evaluate algorithms.

Keywords: Tracking, State Estimation, Metrics, Open Source, Software Design, Software Development

1. INTRODUCTION

Multi-sensor multi-target tracking and state estimation technologies are critical components of maintaining situational awareness (SA) for both military and commercial applications. The ability to detect entities and follow them unambiguously through the state-space is critical in both traditional domains (e.g. air surveillance, maritime SA, ground moving target tracking, computer vision) as well as less conventional domains (e.g. tracking internet transactions, following changes in sentiment, predicting CBRN dispersion, following disease epidemics). However, practitioners in this area face a number of difficulties due to a lack of standardization in the way that these algorithms are evaluated.

There is a multitude of tracking and state estimation methods; each has its own strengths and deficiencies in terms of accuracy, scalability, computational complexity, etc. Novel algorithms are continuously being developed and claims are often made that variants of existing algorithms perform better than previous iterations. If it is difficult for the skilled practitioner to evaluate which algorithm is best for his or her particular application, it is nigh impossible for those with little experience to make this determination. In order for academics to assess the performance of tracking and state estimation algorithms, they require robust implementations of the algorithms being compared; oftentimes this would necessitate implementing someone elses unfamiliar algorithm on their own systems. Also, industrial-scale dataset owners who might not be experts in the mathematics of tracking need to be able to run their data through multiple algorithms quickly in order to compare the performance of the algorithms and select the best algorithm for their application without being influenced or biased by personal favourites.

This paper introduces Stone Soup,[1,2] an open-source framework where researchers and practitioners can implement new tracking and state estimation algorithms for ease of comparison. For instance, users could test a new tracking algorithm against the body of existing algorithms, or they could run real-world or simulated sensor data through multiple tracking algorithms to determine which is best for their application. Although the project is open-source, it can be downloaded and used to implement proprietary algorithms, or to evaluate proprietary/classified data. Stone Soup is designed with modular components, which allows users to rapidly assemble different combinations of components (including in new and unexpected configurations) to build the best tracker for their application. The first open-source version of Stone Soup was released in April 2019.

Stone Soup is initially targeted at two different groups of users. The first is academics conducting research into tracking and state estimation. Academics will be able to implement new tracking algorithms in Stone Soup, evaluate their performance against built-in metrics, and compare the results with those of other tracking algorithms. This will lend credence to claims of improved performance in published research, and facilitate rapid replication by others in the academic community. The second group is the users, owners or processors of real world datasets or sensor systems. Members of this group may have data that they wish to exploit, but they do not necessarily have deep expertise in multiple tracking methods. Stone Soup allows this group to construct the appropriate experiment to test many algorithms against their data to determine which algorithm best suits their application. Stone Soup will provide a quick-start tracker builder that will assist these users in creating experiments without requiring a deep knowledge of the tracker components being used.

2. COLLABORATIVE APPROACH

Most open-source projects are largely self-sustaining once they have been established. However, they require an initial level of functionality around which a user/developer community can coalesce. Therefore, the Stone Soup development process is divided into two phases.

The first phase was the Consortium phase. This phase extended from the Stone Soup program commencement in 2017 until the open source release date in April 2019. In this phase, a consortium of contributors (Defence Science and Technology Laboratory (UK), Defence Research and Development Canada (Canada), Air Force Research Laboratory (US), and the Univeristy of Liverpool (UK)) came together to define the requirements and develop the Stone Soup framework. Work in this stage was focused on building the framework and the components for a few exemplar tracking scenarios; work on additional components was left primarily until the second phase.

The second phase of the Stone Soup project is the Open phase, which began in April 2019 when Stone Soup was open-sourced to the tracking community as the beta version (V0.1b1). During this phase, we expect that the main contributors to the project will be academic researchers who contribute components to the project in order to receive academic recognition. These contributions will be reviewed by the Stone Soup development committee (consisting of members from the UK Defence Science and Technology Laboratory (Dstl) and Defence Research and Development Canada (DRDC)) for merging into the main Stone Soup branch; however, individual researchers may also wish to maintain their own custom Stone Soup code repositories for their own projects.

3. FRAMEWORK ARCHITECTURE

The philosophy behind Stone Soup is to develop an open-source framework with modular, interchangeable components to enable the rapid prototyping and testing of tracking and state estimation algorithms. It will enable users to test, verify, and benchmark a variety of multi-sensor and multi-object estimation algorithms. The Stone Soup framework is built in Python and it will eventually support the prototyping of tracking algorithms in other high-level languages such as Matlab as well as compiled languages such as C and C++. In order to achieve these goals, Stone Soup consists of a framework with modules related to Data, Algorithms, Metrics, Simulators, and Sensor Models (Figure 1).

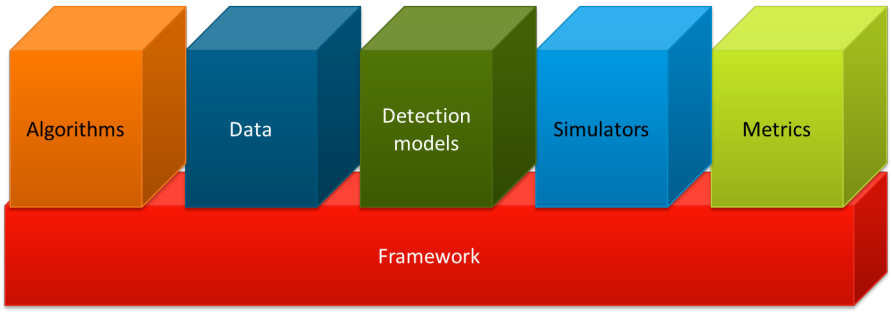

Figure 1. Stone Soup framework architecture.

3.1 Design Philosophy

The central theme of the Stone Soup design philosophy is interchangeability. The framework is designed with the idea that users can easily insert their own components (e.g. predictors, updaters, associators, data readers, etc.) into existing Tracker constructs, and that they could mix and match components in new and unexpected ways. In support of this goal, the Stone Soup code architecture has been built on the principles of modularity and uniformity of external interfaces.

Stone Soup is object oriented and makes use of encapsulation, abstraction and inheritance. Stone Soup trackers are built as hierarchical objects. For example, a Tracker object may contain Track Initiator, Track Deleter, Detector, DataAssociator, and Updater objects (Figure 3). Each of these objects is defined by an abstract class that specifies the external interface for that class; that is, the parameters and functions an object of that class must make available to the outside world (abstraction). As an example, the Updater abstract class specifies that an Updater object must have a measurement_model Property, and that it must have a function update() that returns a State object. Therefore, all implementations of Updaters in Stone Soup (KalmanUpdater, ExtendedKalmanUpdater, ParticleUpdater, etc.) must have the specified Properties and functions (inheritance). This approach ensures that different Updaters are interchangeable (within limits), and that the Tracker can utilize them without knowing the details of the Updater implementation (encapsulation).

In order to support this modularity and common external interfaces, Stone Soup objects are built based on a class inheritance hierarchy (Figure 2). Stone Soup contains a Base abstract class, which is an empty class, and all other Stone Soup classes are descended from this. All implementations of a given abstract class (e.g. Updater) must be children of that abstract class, which defines the common external interface for that abstract class. All implementatons of a given abstract class (e.g. Updater) will perform similar functions. Whenever a new object (e.g. KalmanUpdater) is instantiated, Stone Soup will verify that the instantiation call includes all of the Properties that are required by that class and all ancestors in the hierarchy; this check helps enforce the interchangeability of the Stone Soup modules.

It should be noted the partial inheritance hierarchy in Figure 2 has been designed to be simple and easy-to-understand for new users, and it is not definitive; contributors to Stone Soup are free to suggest changes and develop alternative abstract classes and inheritance hierarchies.

3.2 Components

3.2.1 Data

The basis of most tracking and state estimation is sensor detection data (with the notable exception of Track Before Detect). Stone Soup can generate simulated detection data or it can ingest real-life data from arbitrary real-world sources. Both options offer advantages to potential Stone Soup users. Simulated data is useful for the rapid implementation and testing of new tracking algorithms. Simulated Stone Soup data gives access to the ground truth as well as the sensor output. This is useful because many metrics in Stone Soup require ground truth in order to evaluate the accuracy of the tracking algorithms. Real-life data is useful for testing tracking algorithms against real-world situations (as opposed to laboratory situations) in order to evaluate which

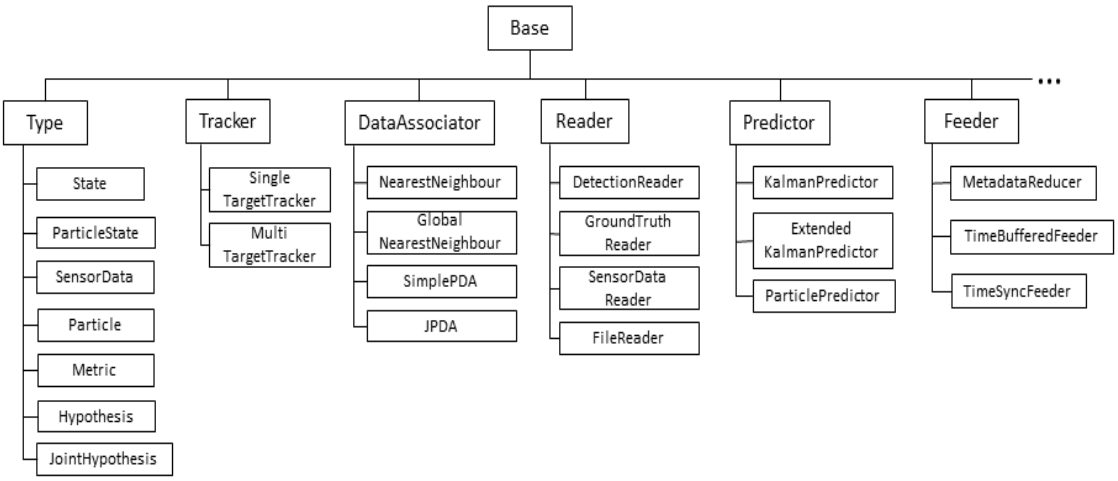

Figure 2. An incomplete view of the Stone Soup class inheritance hierarchy.

algorithm performs the best in a particular situation; however, it will typically not come with the ground truth data required by certain performance metrics.

Stone Soup contains several models that can be used to simulate target (e.g. airplane, ship) movement in order to generate simulated datasets. This simulated data represents the ground truth data, which is later detected by detection models; these models add uncertainty to the measurements representing the uncertainty associated with real sensor measurements. This simulated motion is generated by a number of 1-dimensional motion models, which can be combined into 2- or 3-dimensional transition models that describe the targets movement. The process for generating simulated data is described in more detail in Section 3.2.4. The set of motion models currently implemented in Stone Soup is limited; we hope that users will bring additional motion model implementations to the project.

Stone Soup provides options for writing simulated detection data to an output file, and reading this data back into Stone Soup. This is useful because the simulated data is generated using a Python generator, which generates data just-in-time rather than pre-generating it. This approach saves computer memory space for large simulated datasets, but it also means that, by default, the dataset does not persist. For users who wish to save the simulated data (e.g. to run the same simulated dataset against multiple tracking algorithms), these file writing/reading options provide persistent access to the data. The class ***YAMLWriter*** writes simulated data to an output file in the YAML format, and the class ***YAMLReader*** reads this data back into Stone Soup format.

Stone Soup has been developed to support data from many types of real-world sensors. Working with this data typically requires a ***Reader*** class that reads sensor data from an input stream. It also often requires a ***Feeder*** class that translates data from the sensors measurement space into Stone Soups state space, orders out-of-sequence data chronologically, and removes duplicate data. Stone Soup provides several reader and feeder classes, but users of Stone Soup can easily write their own to serve their specific applications.

3.2.2 Tracking Algorithms

The core of Stone Soup is the tracking functionality. A ***Tracker*** is an algorithm that utilizes several other algorithmic components that work together to associate Detections into Tracks. Stone Soup has been designed to make these components as modular as possible, allowing users to interchange them to evaluate new components and test different combinations. Figure 3 shows the types of these components and the dependency relationships between them. The top-level tracking component is the ***Tracker***; this component associates Detections into Tracks. It requires ***Initiator***, ***Deleter***, ***Detector***, ***DataAssociator***, and ***Updater*** components. Different Trackers can handle a single or multiple targets moving through the space, and can identify and discard clutter.

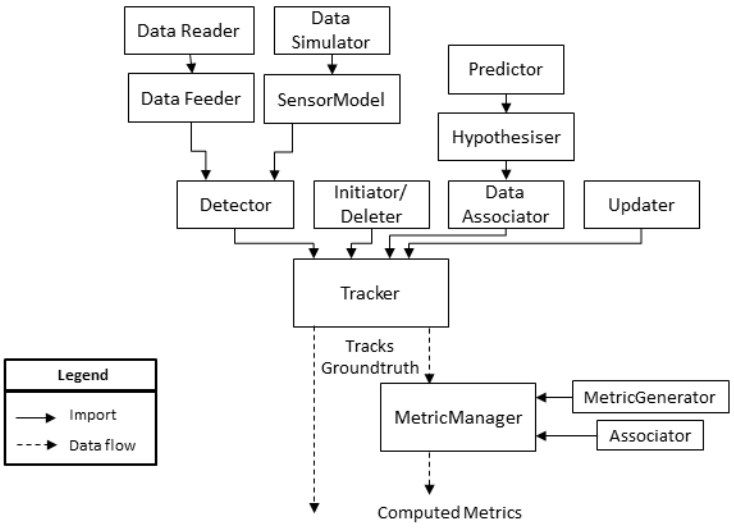

Figure 3. Stone Soup module structure and information flow.

The ***Initiator*** initializes a Track item from a Detection that is determined not to be part of another Track; it sets up the initial parameters that allow the Track position to be predicted into the future. The ***Deleter*** kills a Track after certain conditions have been met which indicate that the track has been lost (e.g. no Detections matching the Track prediction have been observed for a certain number of time steps, or the covariance matrix of the prediction is very large for the same reason).

The ***Detector*** provides a stream of Detections to the Tracker. The data source for the Detector is a detection simulator (for simulated data), real data read from a file or a data stream, or any other data input methods that users would like to develop. In order to use simulated data, the user may employ a ground truth data simulator that utilizes a transition model to model the motion of the targets. The simulated ground truth generated by this simulator is then processed by detection simulator, which uses a measurement model that approximates how a sensor detects the ground truth data (with uncertainty of detection location, missed detections, etc.). As discussed in Section 3.2.3, both the ground truth and the Detections generated by the measurement model are used as inputs to the various Stone Soup metrics.

The ***Predictor*** and ***Updater*** are the foundation of any tracking scheme; they implement the functionality of the underlying basic filter. While Stone Soup currently contains implementations of a few basic filters, this is an area with rich opportunities for development by the academic/user community. Typically, a Predictor and Updater with matching approaches must be paired (e.g. both Kalman, both Particle).

The data association function of a tracker with which the tracking community is likely to be familiar is encapsultated in Stone Soup using two different components: the ***Hypothesiser*** and the ***Data Associator***. Roughly speaking, the data association process associates Detections with Tracks (Targets). In Stone Soup, the ***Hypothesiser*** calculates the measures (e.g. distance, probability) between the Detections and predicted Track states on which this decision is based, and the ***Data Associator*** makes this decision. We realize that different classes of tracking algorithms calculate and represent the state of the system in different ways. As befits a project that is under continuous development, accomodations for this fact are ongoing. Please refer to Section 5 for future plans in this area. Users who are implementing new tracking algorithms in Stone Soup are recommended to look at existing implementations to help them decide the best approach to implementing their own algorithm.

3.2.3 Metrics

One of the main goals of Stone Soup is to compare the performance of different tracking algorithms. Currently, Stone Soup provides support for metrics that evaluate the accuracy of tracking algorithms. These metrics accept

the output of a Tracker operating over a dataset and evaluate how well the Tracks (what we think happened) match the ground truth (what actually happened).

There are three major components related to Stone Soup metrics (Figure 3). The ***MetricManager*** parses and stores the Tracks and ground truth produced by the Tracker (and any other outputs produced by future Tracker implementations), and calls the metrics that have been assigned to it. Several of the metrics for Tracker performance measure how well the identified Tracks match the ground truth. In order to measure this, Stone Soup uses an ***Associator*** to associate Tracks and ground truth. The metrics themselves are calculated by ***MetricGenerators***. These classes accept the Tracker outputs and return the calculated metrics. Stone Soup provides implementations of several industry-standard metrics; users are also free to implement other metric suites and add them to the Stone Soup code base.

3.2.4 Data Simulators

In the case where Stone Soup users do not have real datasets available to test tracking algorithms, Stone Soup provides simulators that can generate realistic Detection datasets. Simulating Detections is a 2-step process: 1) generate ground truth data based on transition models, and 2) model the sensor state and behaviour as it detects the ground truth Targets.

The Stone Soup ***GroundTruthSimulators*** generate data related to one or more Targets moving through the detection area. They generate position data at defined time steps, and the Target motion is based on user-selected transition models. The ***DetectionSimulators*** simulate detections by a sensor platform whose characteristics are specified by the user-specified measurement model. The simulator accepts ground truth data as inputs, and outputs Detections as well as Clutter measurements based on the state of the measurement model. Detections can be generated using a generic sensor model or one of the sensor models discussed in Section 3.2.5.

Simulated data is important for fully exploring the capabilities of different tracking algorithms because ground truth data is needed for several of the the Stone Soup metrics and real sensor data often does not include ground truth data.

3.2.5 Sensor Models

Stones Soup contains a ***SensorPlatform*** class that represents a mobile platform that can move in 2D or 3D space to represent the motion of an aircraft, ship, or other mobile sensor. The platform motion can consist of any of the motion models that have been previously described. A platform can contain multiple sensors to model a variety of real-world platforms; e.g. a platform containing both an EO/IR sensor and a radar sensor. The sensors can be mounted at arbitrary positions on the platform.

Stone Soup contains a generic sensor model as well as several sensor models that simulate the physics of sensor detections in greater detail. These sensor models replicate the characteristics, strengths, and weaknesses of distinctive real-world sensors so users can analyse the performance of different tracking algorithms against sensor-specific datasets.

Each sensor on a platform contains an instance of a measurement model. The measurement model is responsible for transforming the models state vector into a measurement vector. The measurement models also specify the noise properties of the measurement; e.g. noise covariance, so that these can be used in the tracking filter to update the state estimates. The measurement model can also be used to generate a set of noisy measurements drawn from a Gaussian distribution; these measurments can be used in the particle filter implementations.

The sensor model area is a rich area for contribution by academics in the StoneSoup user community.

3.2.6 Stone Soup Component Implementations

The objective of the Stone Soup development team prior to the April 2019 open source release date was to develop a framework to which the tracking community can contribute their own implementations of different algorithms. However, as part of the initial development, the Stone Soup development team implemented a few tracking algorithms along with their component parts as a starting point. We expect that these implementations will serve as a base case that other contributors can build upon and make their own contributions to the project. A listing of the beta release components can be found in Table 1.

Simulator
SingleTargetGroundTruthSimulator
MultiTargetGroundTruthSimulator
SimpleDetectionSimulator

File Reader
JSON_AISDetectionReader
CSVDetectionReader
YAMLReader

File Writer
YAMLWriter

Feeder
MetadataReducer
TimeBufferedFeeder
TimeSyncFeeder

Transition Models
CombinedLinearGaussianTransitionModel
LinearGaussianTimeInvariantTransitionModel
ConstantVelocity
ConstantAcceleration
Singer
SingerApproximate
ConstantTurn

Measurement Models
LinearGaussian
RangeBearingElevationGaussianToCartesian
RangeBearingGaussianToCartesian
BearingElevationGaussianToCartesian

Sensor models
RadarRangeBearing
RadarRotatingRangeBearing

(mobile) Platform models
SensorPlatform

Initiator
SinglePointInitiator
LinearMeasurementInitiator
GaussianParticleInitiator

Data Associator (metrics)
EuclideanTrackToTrack
EuclideanTrackToTruth

Deleter
CovarianceBasedDeleter
UpdateTimeStepsDeleter
UpdateTimeDeleter

Predictor
KalmanPredictor
ExtendedKalmanPredictor
UnscentedKalmanPredictor
ParticlePredictor

Updater
KalmanUpdater
ExtendedKalmanUpdater
UnscentedKalmanUpdater
ParticleUpdater

Hypothesiser
DistanceHypothesiser
FilteredDetectionsHypothesiser
PDAHypothesiser

Data Associator (tracker)
NearestNeighbour
GlobalNearestNeighbour
SimplePDA
JPDA

Tracker
SingleTargetTracker
MultiTargetTracker
MultiTargetMixtureTracker

Metric Manager
SimpleManager

Metric Generator
BasicMetrics
GOSPAMetric[3]
OSPAMetric[4]
TwoDPlotter
SIAPMetrics[5]

Measure
Euclidean
EuclideanWeighted
Mahalanobis
SquaredGaussianHellinger
GaussianHellinger

Table 1: Component implementations in Stone Soup (April 2019).

4. USING STONE SOUP

4.1 Run Manager

The process for building a Stone Soup Tracker to run an experiment can be complex and time-consuming if performed manually (see supplemental materials associated with Section 4.3). We expect that members of the Stone Soup user community will want to run large blocks of experiments. They may wish to run experiments on a given Tracker with certain components swapped or parameters varied over a range in order to find the optimal combinations. Alternatively, they may wish to run a Monte Carlo simulation that runs the same experiment many times with different input data in order to see how the Tracker performs over a wide variety of situations. In order to reduce the workload for building and running these types of experiments, Stone Soup provides a Run Manager (available a few weeks after the initial beta release) that can automatically build, run, and collect the results from a large block of experiments.

The Tracker and Metrics configurations to be used in an experiment are encoded into an experiment configuration file, along with the conditions for running the experiment multiple times or varying one or more of the components or model parameters. This experiment configuration file is then fed into the Run Manager, which can build and run all of the Trackers specified. The Run Manager then collects the results and metrics from the experiments and then exports them to an output file for further processing and analysis.

4.2 Jupyter Notebooks

Stone Soup comes with extensive documentation regarding its component parts and how they fit together to build Tracker experiments. However, sometimes the best way to learn is through example and demonstration. The Stone Soup development team makes extensive use of Jupyter Notebooks[6] which combine Python code with supplemental explanations into a format that can be executed using a web browser as a graphical frontend. The Stone Soup beta release provides a library of Jupyter Notebooks (see Section 7) that demonstrate different use cases and show how Stone Soup components fit together to build Tracker experiments. Members of the Stone Soup user community can experiment with these Notebooks in order to learn how Stone Soup works, and they can modify the code in order to integrate their own components developed for use in the Stone Soup framework, or to import their own datasets.

4.3 Use Case

The Jupyter Notebook One of the Jupyter Notebooks in the Stone Soup library (see Section 7) entitled "Stone Soup SPIE Use Case.ipynb" demonstrates a simple use case for the Stone Soup framework. This use case showcases the different components used to build a Stone Soup Tracker and how they fit together. In this example (Figure 4), a Tracker operates over a simulated dataset, and metrics are calculated indicating how well the Tracker performed.

The Tracker used in this scenario is a Kalman Filter-based Tracker. It uses basic Stone Soup implementations such as the Nearest Neighbour data associator, Mahalanobis Distance hypothesiser, track initiator, and data simulator, as well as the Kalman Predictor and Updater. The data simulator generates ground truth Target location data and detections from the simulated sensor; the Tracker consumes this data and then outputs the Tracks calculated by the Tracker. The results are processed by a MetricManager; it uses several MetricGenerators to calculate metrics over the calculated Tracks, as well as a Track to Truth data associator that measures how well the calculated Tracks match the ground truth.

Figure 5 shows the graphical representation of the Tracks produced by a particular run of this Use Case. Table 2 shows the numerical results of the Basic Metrics and the SIAP Metrics calculated by this run, and Figure 6 shows the OSPA distance calculated by the OSPA Metrics at each time step over the course of the experiment.

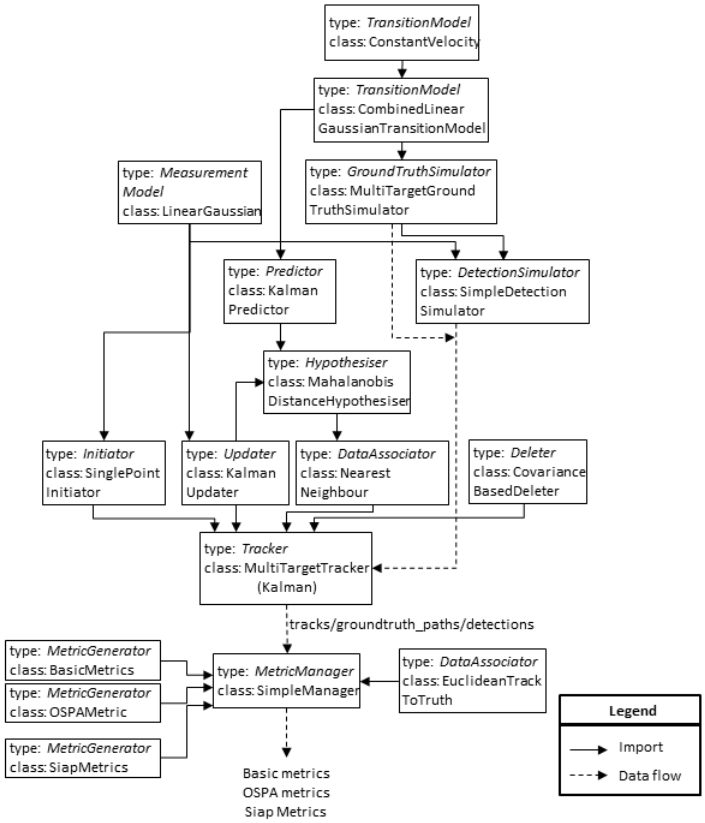

Figure 4. Architecture of Stone Soup Use Case.

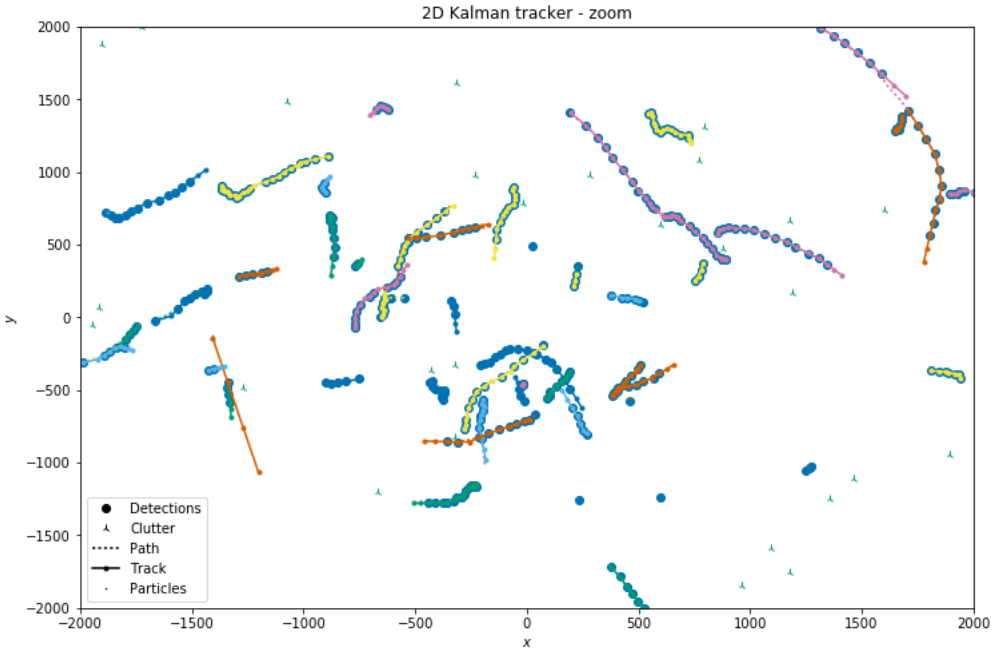

Figure 5. Graphical output of Kalman MultiTargetTracker.

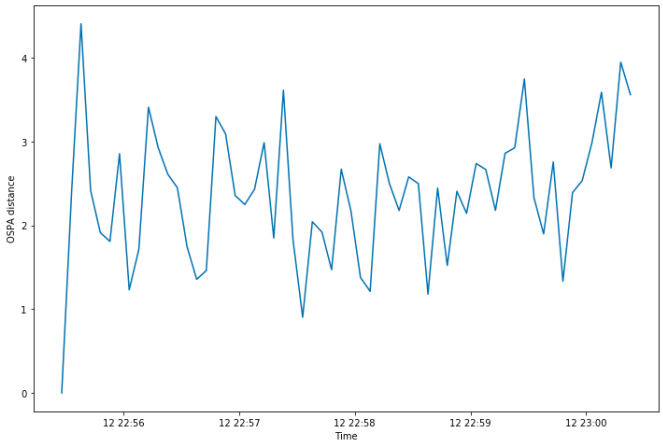

Figure 6. OSPA distance metric for a particular run of the Use Case over the time span of the experiment.

Metric	Use Case Value
Number of Tracks	63
Number of Targets	55
Track-to-Target ratio	1.145456
SIAP A	1.0
SIAP C	0.963925
SIAP LS	0.898119
SIAP LT	1527.50
SIAP S	0.118734

Table 2. Metrics for a particular run of the Use Case.

5. FUTURE WORK

Following the Stone Soup open-source release in April, 2019, we anticipate that members of the tracking community will begin to contribute implementations of trackers and various components to the project. One focused mechanism for this will be the working group established under the International Society for Information Fusion (ISIF), called the Open Source Tracking and Estimation Working Group (OSTEWG) (https://isif-ostewg.org/). This group aims to form a community focused on development and use of algorithms in Stone Soup for an annual use/data-case and to present the results at a Special Session at each ISIF Fusion Conference.

In addition to this, the current Stone Soup development team has plans for a few critical additions. Currently, the primary means of building and running Stone Soup experiments is through Jupyter Notebooks. We expect that this will be a natural and comfortable interface for academic researchers; however, this interface will not meet the needs of all potential Stone Soup users. Members of the defence or security community who bring large sensor datasets to Stone Soup may not have a background in tracking or software development, so they will need a simpler way to build and run experiments. A major planned addition to Stone Soup moving forward will be the User Interface (UI). This will be developed as a web browser-based Graphical UI that can be used to interact with Stone Soup. The UI will have three main components. The Experiment Builder will allow users to select Stone Soup components to build Trackers and experiments. The Builder will also provide a wizard to guide users with minimal experience in tracking through the process of building an experiment. The Run Manager section of the UI will interact with the Stone Soup ***RunManager*** to run experiments and collect the output Tracks and metrics. The Results Viewer will allow users to view and interact with the results of the experiments. It will show plots of the Detections/Tracks generated by the experiment, it will display metrics results, and it will contain a Playback Viewer that will allow the user to step through the experiment by timestep. This UI will be one of the major efforts of the Stone Soup development team moving forward.

Most of the tracking components currently implemented in Stone Soup deal with radar-type sensor detection data. Another area in the tracking space that has not yet been addressed is Wide-Angle Motion Imagery (WAMI). WAMI systems contain multiple high-pixel cameras that perform surveillance on a large area (tens or hundreds of square kilometres) and then stitch the images together in order to track and analyse the behaviour of a large number of individuals, vehicles, etc. WAMI systems provide much richer intelligence than single camera platforms with a narrower field of view. Moving forward, we plan to add support for WAMI tracking into Stone Soup.

The Stone Soup architecture is currently oriented towards tracking algorithms that represent the state of the system as a set of Tracks, where new Detections are associated to a single Track. However, some tracking algorithms do not maintain distinct Tracks but instead represent the state of the system as a Gaussian Mixture (e.g. the Gaussian Mixture Probabiliistic Hypothesis Density (GMPHD) filter). We are currently modifying

the Stone Soup architecture to better accomodate these types of algorithms, and we expect these changes to be merged into the Stone Soup code base in the near future. Once these modifications have been made, we will also make available our implementation of the GMPHD filter.

The tracking academic community has developed many variants of the main tracking algorithms. There are tweaks to algorithms to improve their accuracy in certain situations, and there are upgrades to improve the running time of the algorithms. As an example, one of the tracking algorithms currently implemented in Stone Soup is the Joint Probabilistic Data Association Filter (JPDAF). As part of its operation, JPDAF calculates the probabilities related to every permutation of associations between Detections and Tracks. In a Target-dense environment, an enumerative approach to this calculation will soon fall prey to combinatorial explosion. Different researchers have developed approaches to performing these calculations in less than exponential time.[7] Part of the future work on Stone Soup will be to implement select variants of tracking algorithms to demonstrate these types of gains in processing speed.

6. CONCLUSION

Tracking is an important technology with many military, security, and commercial applications. However, the tracking community can be fragmented when it comes to sharing ideas. We saw that there was no standard method for academics to analyse new tracking algorithms or benchmark them against industry-standard algorithms. Neither was there a convenient capability for owners of sensor datasets to experiment with applying different tracking algorithms to their datasets. In response to these needs, we have developed the Stone Soup tracking and state estimation framework, which is open source and open for public use by the tracking community. Stone Soup will serve as a testbed for new algorithms developed by the academic community. It will enable academics to share new algorithm implementations in a form that is easily verifiable and reproducible, and it will allow for convenient performance comparisons. Stone Soup will also allow sensor dataset owners to quickly evaluate multiple tracking algorithms against their datasets and compare the performance of different algorithms. It will support this analysis free from bias towards personal favourites. The Stone Soup framework and several tracking algorithm implementations have been developed by a core team of developers from the United Kingdom, Canada, and the United States. It is now open to the research and operator community so that users can contribute new tracking algorithm implementations and apply Stone Soup to their individual use cases. It is our belief that Stone Soup will serve as a valuable tool for standardizing the development and evaluation of tracking algorithms.

7. SUPPLEMENTARY MATERIALS

The Stone Soup open source code repository is located at https://github.com/dstl/Stone-Soup.

A library of Jupyter Notebooks that demonstrate the various aspects of Stone Soup functionality is located at https://github.com/dstl/Stone-Soup-Notebooks.

Supplementary data sets that can be used with the Jupter Notebooks are located at https://isif-ostewg.org/data.

The authors would like to acknowledge other non-author members of the development team: Charles England, Samuel Thomas, Mark Campbell, Dr. Bhashyam Balaji, Sarah Babbitt, Professor Simon Maskell, Paul Horridge, Yifan Zhou, James Wright, and Stephen Ablett.

REFERENCES

[1] Thomas, P. A., Barr, J., Balaji, B., and White, K., "An open source framework for tracking and state estimation (stone soup)," in [*Signal Processing, Sensor/Information Fusion, and Target Recognition XXVI*], **10200**, 1020008, International Society for Optics and Photonics (2017).

[2] Thomas, P., Barr, J., Hiscocks, S., England, C., Maskell, S., Balaji, B., and Williams, J., "Stone soup: An open-source framework for tracking and state estimation," *ISIF Perspectives on Information Fusion* , 14–19 (March 2019).

[3] Rahmathullah, A. S., Garca-Fernndez, . F., and Svensson, L., "Generalized optimal sub-pattern assignment metric," in [*2017 20th International Conference on Information Fusion (Fusion)*], 1–8 (July 2017).

[4] Schuhmacher, D., Vo, B., and Vo, B., "A consistent metric for performance evaluation of multi-object filters," *IEEE Transactions on Signal Processing* **56**, 3447–3457 (August 2008).

[5] S. J. Karoly, J. W. Wilson, H. D. and Maluda, J. W., "Single Integrated Air Picture (SIAP) Attributes Version 2.0," Tech. Rep. Technical report ref. SIAP SE TF-TR-2003-029, Joint Single Integrated Air Picture (SIAP) System Engineering Task Force – Arlington, VA 22203 (August 2003).

[6] Project Jupyter, "Jupyter." https://jupyter.org/. Accessed: 2019-03-14.

[7] Crouse, D. F. and Willett, P., "Computation of target-measurement association probabilities using the matrix permanent," *IEEE Transactions on Aerospace and Electronic Systems* **53**, 698–702 (April 2017).