Stone Soup: No Longer Just an Appetiser

Steven Hiscocks, Jordi Barr, Nicola Perree,
James Wright, Henry Pritchett, Oliver Rosoman,
Michael Harris, Roisín Gorman, Sam Pike
Cyber and Information Systems
Defence Science and Technology Laboratory
Salisbury, UK
stonesoup@dstl.gov.uk

Peter Carniglia
Radar Sensing and Exploitation
Defence Research and Development Canada
Ottawa, Canada
peter.carniglia@drdc-rddc.gc.ca

Lyudmil Vladimirov, Benedict Oakes
{*Electrical Engineering & Electronics, CDT in Distributed Algorithms*}
University of Liverpool
Liverpool, UK
{l.vladimirov,sgboakes}@liverpool.ac.uk

***Abstract*—This paper announces version 1.0 of Stone Soup: the open-source tracking and state estimation framework. We highlight key elements of the framework and outline example applications and community activities.**

Stone Soup is engineered with modularity and encapsulation at its heart. This means that its many components can be put together in any number of ways to build, compare, and assure almost any type of multi-target tracking and fusion algorithm. Since its inception in 2017, it has aimed to provide the target tracking and state estimation community with an open, easy-to-deploy framework to develop and assess the performance of different types of trackers. Now, through repeated application in many use cases, implementation of a wide variety of algorithms, multiple *beta* releases, and contributions from the community, the framework has reached a stable point.

In announcing this release, we hope to encourage additional adoption and further contributions to the toolkit. We also acknowledge and express appreciation for the many contributions of time and expertise donated by the tracking and fusion community.

***Index Terms*—tracking, fusion, open source, framework**

I. Introduction

Stone Soup [1] is a flexible, modular, open-source framework for developing and proving a wide variety of tracking and information-fusion-based solutions. Its modularity provides components which can be put together in many ways, allowing any number of fusion, tracking and sensor management algorithms to be built. Individual components function analogously to those in a machine, being designed to be interchangeable, encapsulated and capable of deployment in a multitude of configurations. Stone Soup has, to a degree, derived motivation from the success of similar communities for computer vision, machine learning, and data science.

In order to satisfy the diverse needs of the tracking and state estimation community, Stone Soup has been designed with at least three separate groups in mind. Firstly, *algorithm developers* must be able to create novel methods for target tracking and state estimation; secondly, *algorithm users* should have confidence that their methods are fit for purpose, metricated and deliver the promised improvements; and finally, *system engineers* require methods that deliver results appropriate for their intended use which work within the requirements of new or existing systems. In all cases the properties of encapsulation and modularity promote efficiency and reuse.

Open source principles are key for Stone Soup. To maximize collaboration across academia, industry, and government, permissive licensing is mandated for core components and encouraged for all contributions. Notwithstanding the breadth of contributors, software integration rigour is employed. Stone Soup is open-source and version-controlled on GitHub.[1] Contributions are subject to testing and review prior to inclusion on the *main* branch. Documentation is extensive and includes tutorials, example applications, and demonstrations using data sources.[2]

The Stone Soup concept dates from 2014 [2]. Development started in earnest in 2017 [3], with the *alpha* release demonstrated at a plenary session during the 2018 Fusion conference in Cambridge, UK [4]. This *alpha* demonstration compared an extended Kalman filter and particle filter in a multi-target tracking simulation, utilising the modularity, encapsulation and inheritance provided by Stone Soup's object orientation. This was followed, after much development work, by the full open sourcing of the *beta* release in April 2019 [5] which expanded the suite of components to include (joint) probabilistic data association, an unscented Kalman filter, simulated sensors, and metric capability. Stone Soup has been applied to a plethora of application areas, including video tracking, drone tracking, platform/sensor simulation, sensor management and more (see e.g. [6]).

While it has grown to maturity, Stone Soup has benefited from the support of a number of communities. In its early phase, it was developed by The Technical Cooperation Pro-

[1] https://github.com/dstl/Stone-Soup
[2] https://stonesoup.rtfd.io

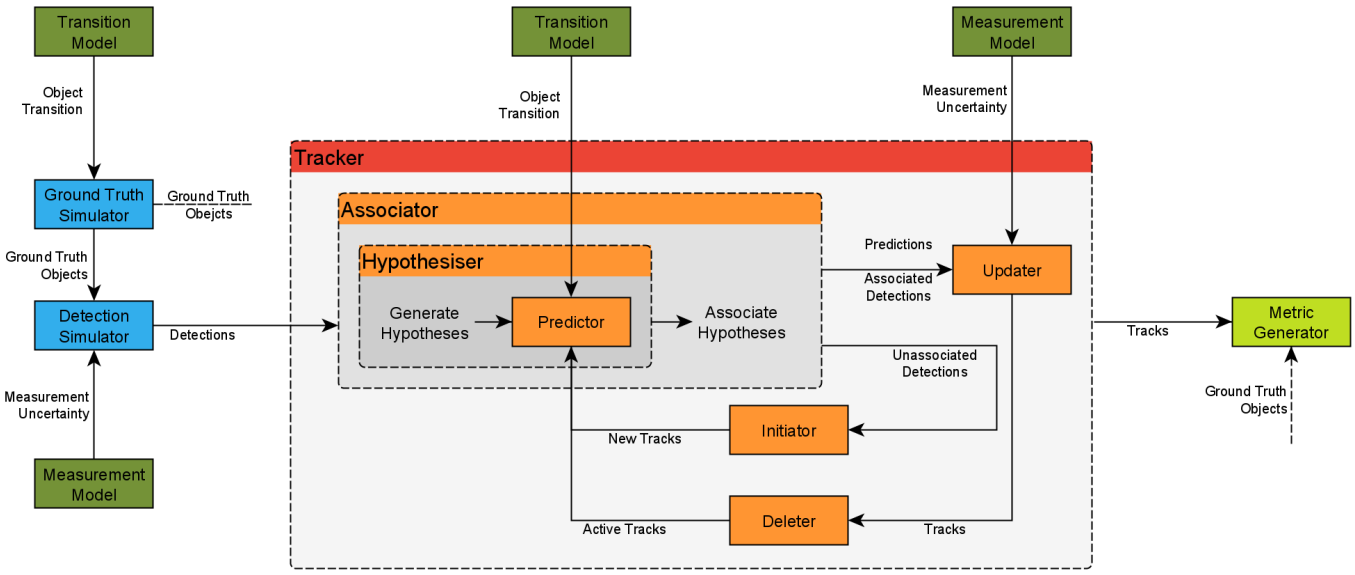

Fig. 1. Example multi target tracker overview, with ground truth and detection simulation, and metrics generation.

gram (TTCP).[3] From 2017, it was the chosen means of collaboration for the incipient Open-Source Tracking and Estimation Working Group (OSTEWG), a working group supporting the International Society of Information Fusion (ISIF) mission to enable collaboration among active information fusion researchers.

Stone Soup is now released as version 1.0 which indicates stability in the component interfaces and reliability for inclusion as a dependency in other software development projects, limiting breaking changes only to major version increments. This represents the culmination of a number of years of effort and provides the tracking community with a robust and expanding series of tools to enhance and expand the reach of their work.

This paper is structured as follows. Section II describes the framework itself with design principles and details on important components. Simulation and experimentation are covered in §III and a selection of applications and general community activities are detailed in §IV before we summarise and look to the future.

II. Framework & Components

Python is the primary language of the framework, offering as advantage its ubiquity within the data science community, its openness, flexibility, and its popularity in a large number of education environments. The framework uses the highly optimised NumPy [7] and SciPy [8] libraries for linear algebra, statistical functions and optimisation.

Exploiting the advantages of object-oriented programming, a key aspect to adding algorithms to the framework is successfully breaking the algorithms down to the sub-components. This helps users to reduce effort in adding new algorithms, allows rapid prototyping of components and enables putting together components in novel combinations.

Documentation has been a focus during development to record the framework interfaces, components and features, built utilising Sphinx [9] documentation library. In addition, Sphinx-Gallery [10] has been used to build tutorial, examples and demonstrations directly from code, to ensure these examples are always functional, and to provide a form of integration testing.

All contributions to Stone Soup are publicly peer reviewed, allowing anyone to comment. This ensures that contributions to the framework are correctly implemented, as well as being tested and documented appropriately.

In any software development project, tests are critical to ensure individual units function as expected and integrate as a whole. The Pytest [11] testing framework is used mainly for unit tests, and integration testing includes documentation generation via Sphinx-Gallery [10].

To enable ease of integration of additional components, a plugin system has been added to the framework. This enables bespoke, proprietary or niche components to be maintained and version controlled separately, whether closed or open source, to the main framework.

Figure 1 shows one example configuration of classes for a multi target tracker, which includes simulated targets and detections, utilising transition and measurement models. This configuration can be used with different models, predictors, updaters, etc. and compared using selected metrics of interest.

A. Models

The Stone Soup library contains standard models to describe the evolution over time of a variety of targets (e.g. space,

[3]The Technical Cooperation Program (TTCP) is a long-standing international organisation concerned with cooperation on defence science and technology matters. It's membership comprises Australia, Canada, New Zealand, the United Kingdom (UK) and the United States of America (USA).

air, ground, maritime) for use in filtering applications. In Stone Soup these types of models are called *transition models*, though more broadly in the literature are referred to as state, state-transition, or state-space models. In addition to transition models, *measurement models* have been implemented to represent the conversion from state space to the sensor-specific measurement space. Measurement models may also be referred to as observation models.

Most generally, the transition and measurement models are represented mathematically as

$$\mathbf{x}_{k|k-1} = f(\mathbf{x}_{k-1}, \mathbf{w}_k, \mathbf{u}_k) \quad (1)$$

$$\mathbf{z}_k = h(\mathbf{x}_k, \mathbf{v}_k) \quad (2)$$

where $\mathbf{w}_k$ and $\mathbf{v}_k$ are the transition and measurement noises, respectively. $\mathbf{u}_k$ represents a control input vector. Stone Soup contains over twenty-five transition and measurement models built-in, focused on both linear and non-linear kinematic state estimation. The documentation also provides guidance for users to allow them to implement their own models.

1) Transition Models: In the Stone Soup framework, transition models described by Equation 1 can be assembled in parts, with each independent dimension being defined by a separate model. The general operation of a transition model in Stone Soup is demonstrated in Figure 2. Note that the implementation of transition models is independent of usage – while currently implemented models are focused primarily on kinematic applications, the framework supports any model described by Eq. 1 so long as it follows the template in Figure 2

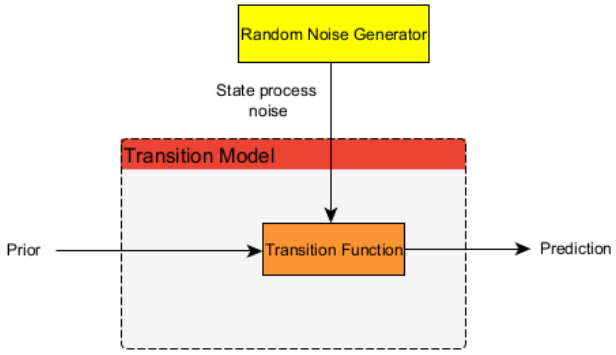

Fig. 2. The transition model structure which implements Equation 1

2) Measurement Models: Measurement models (i.e. Eq. 2) within Stone Soup are used to represent the mapping from the target's state space to the sensor observation space. As such, unique measurement models must be used for each combination of measurement coordinate system to state coordinate system. While the measurements are sensor specific, they are not necessarily sensor exclusive; more than one type of sensor can return the same types of measurements (Ex. range, azimuth, elevation). The general operation of a measurement model in Stone Soup is presented in Figure 3.

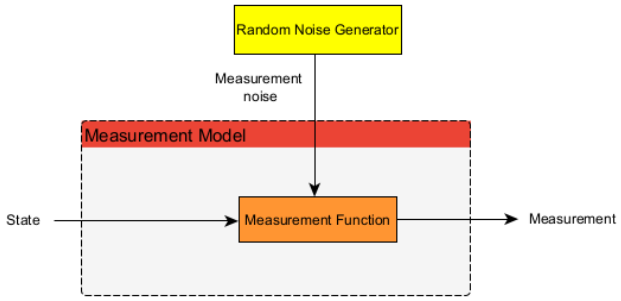

Fig. 3. The composition of the measurement model used to implement Equation 2

B. Filtering

Filtering of both single and multi target densities in Stone Soup is broken into two steps: prediction and update. Each step is performed by a dedicated component, namely the `Predictor` and `Updater` classes, that form the base classes for concrete algorithmic implementations of each step.

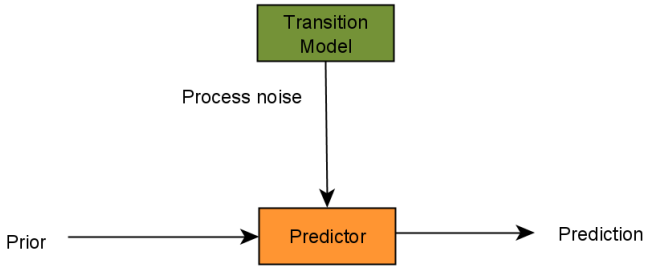

Fig. 4. Predictor implements Equation 3

1) Prediction: The `Predictor` class is responsible for propagating a given state density through time, making use of a specified `TransitionModel`. In mathematical terms, the `Predictor` implements the Chapman-Kolmogorov equation:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \int_{-\infty}^{\infty} p(\mathbf{x}_k|\mathbf{x}_{k-1}) p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1}) d\mathbf{x}_{k-1} \quad (3)$$

where $p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})$ is the posterior state at time $k-1$, $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ is the transition kernel (defined by the chosen `TransitionModel`), and $p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})$ is the generated predictive posterior. The implemented structure is shown in Figure 4.

Stone Soup contains a large collection of predictors, including (but not limited to):

- `KalmanPredictor`: Prediction step of the standard, linear-Gaussian, Kalman filter [12];
- `SqrtKalmanPredictor`: Version of the Kalman prediction that operates on the square root parameterisation of the state [13];
- `InformationKalmanPredictor`: Predictions step using the information form of the Kalman filter [14];

- `Extended/UnscentedKalmanPredictor`: Extended [15] and Unscented [16] Kalman filter prediction for non-linear, Gaussian models;
- `ParticlePredictor`: Prediction step of the (bootstrap) Particle filter [17], [18] for non-linear, non-Gaussian models;
- `ParticleFlowKalmanPredictor`: Prediction step of the stochastic Particle flow filter that uses and underlying Kalman predictor to maintain a state covariance [19];

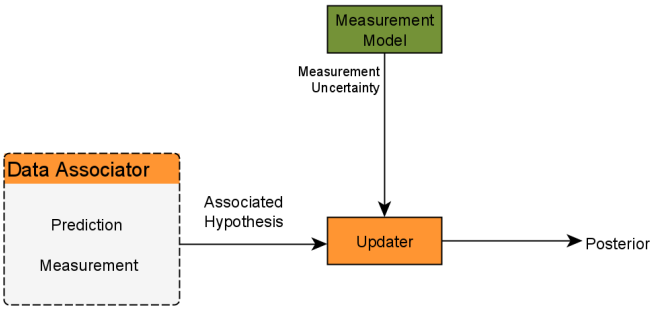

Fig. 5. Updater components in the Stone Soup library. This implements Equation 4

2) Update: The goal of the `Updater` class is to generate the posterior state estimate, based on a predicted state (as computed by a `Predictor`) and a (sequence of) measurements. In doing so, the component makes use of a `MeasurementModel` that is either provided at object instantiation, or comes attached to each measurement.

Mathematically, the above is achieved by applying Bayes rule[4]:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k)} \tag{4}$$

where $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$ is the predictive posterior - computed by the `Predictor` via (3), $p(\mathbf{z}_k|\mathbf{x}_k)$ is the measurement likelihood (as defined by the `MeasurementModel`), $p(\mathbf{z}_k)$ is the evidence, and $p(\mathbf{x}_k|\mathbf{z}_{1:k})$ is the computed posterior state. Figure 5 shows how Stone Soup undertakes this computation.

Some examples of updater implementations included in Stone Soup are:

- `KalmanUpdater`: Update step of the standard, linear-Gaussian, Kalman filter;
- `SqrtKalmanUpdater`: Kalman update variant that operates on the square root parameterisation of the state;
- `InformationKalmanUpdater`: Update step using the information form of the Kalman filter;
- `Extended/UnscentedKalmanUpdater`: Extended/Unscented Kalman filter update step for non-linear, Gaussian models;
- `ParticleUpdater`: Update step of the (bootstrap) Particle filter for non-linear, non-Gaussian models;
- `GromovFlowParticleUpdater`: Update step of Gromov method for stochastic Particle flow filters [20];
- `PHDUpdater`: Implementation of the Gaussian Mixture Probability Hypothesis Density (GM-PHD) [21] update.

[4]The formulation for multi-target/multi-measurement `Updater` implementations may differ (e.g. `PHDUpdater`), but is of similar form.

C. Data Association

The goal of the data association step is to find a mapping,

$$g_k : Z_k \rightarrow X_{k|k-1} \tag{5}$$

between the set of (M) measurements at k, $Z_k = \{\mathbf{z}_1, ..., \mathbf{z}_M\}_k$ and the set of (N) predicted states, $X_{k|k-1} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}_{k|k-1}$. In Stone Soup the `DataAssociator` type produces g_k by generating association hypotheses $\{\mathbf{z}\}_k \in Z_k \rightarrow \{\mathbf{x}\}_{k|k-1} \in X_{k|k-1}$ using a `Hypothesiser` type. This is shown in Figure 6.

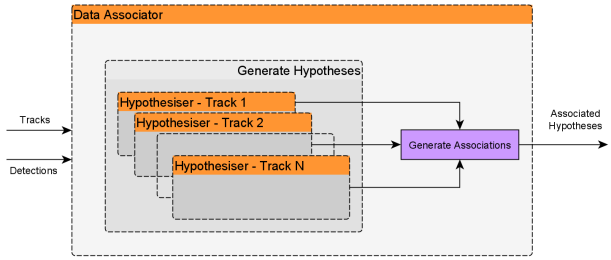

Fig. 6. Data Associator implementation (i.e. Equation 5) in the Stone Soup library.

In order to calculate a posterior estimate via the updater (e.g. Figure 5) an explicit association between prediction and measurement must always be made. In single-target scenarios, a single `Hypothesis` can be used. In situations with clutter or multiple targets, a `DataAssociator` is employed in conjunction with a `Hypothesiser`, which generates sets of detection-to-track/prediction hypotheses. Hypothesisers vary in their complexity and may propose associations based on defined metrics (e.g. `DistanceHypothesiser`) or may make probabilistic associations (`PDAHypothesiser`), or, indeed, associations which include the empty set. Both classes support locally and globally optimised hypothesis pairing (see for example `GlobalNearestNeighbour` and `JPDA`).

The computational speed of data association with any of these algorithms can also be enhanced with a time parameterised R-tree (TPR) [22] and k-d tree [23] mixins. These methods are efficient in more complex tracking scenarios with large numbers of targets, detections, and high clutter density.

D. Sensor Management

In addition to algorithms for tracking and state estimation, Stone Soup also has components for building autonomous sensor management algorithms. Sensor management problems can be considered Partially Observed Markov Decision Processes (POMDPs, e.g. [24], [25]). At k one can use a POMDP to find an optimal set of actions, $\{a\}_k^*$ via a policy,

$$\mu_k^*(X_{k|k-1}) = \underset{\{a\}_k \in \mathcal{P}(A_k)}{\operatorname{argmax}} R_k(X_{k|k-1}, \{a\}_k) \tag{6}$$

where $R_k(\cdot)$ is a (so-called reward) function of some allowable subset of the power set of all sensors' actions $\{a\}_k \in \mathcal{P}(A_k)$

and the predicted set of states $X_{k|k-1}$. These latter are themselves inferred from a sequence of observations $Z_{1:k-1}$ (as e.g. in previous sections). In Stone Soup a `RewardFunction` supplies $R_k(\cdot)$ and the `SensorManager` provides an optimiser which undertakes $\operatorname{argmax} R_k(X_{k|k-1}, \{a\}_k)$. A_k is collected from sensors. This is shown in Figure 7.

Stone Soup contains a number of baseline approaches to sensor management. The classes developed derive from `SensorManager` and `RewardFunction` base classes, and adaptations have been made to sensors which enable them to report available actions and undertake actions tasked by the manager. A sensor manager has knowledge of sensors under its domain, contains a method for evaluating actions based on a reward function, and knowledge of the tracks up to the current time step.

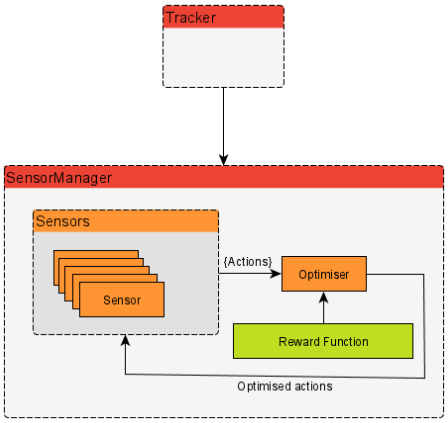

Fig. 7. Sensor Management algorithm structure as implemented in Stone Soup.

Additional methods available in the framework include optimised approaches utilising the SciPy *optimize* library [8], and example reward functions. These components can be used in conjunction with Stone Soup's tracking and simulation capabilities to build a scenario where sensors are being tasked autonomously, which maximises tracking performance over time. The base structure provided also allows users to craft their own sensor management components and integrate them with existing algorithms and performance evaluation metrics. Recent innovations look to employ reinforcement learning methods to optimise sensor allocation.

III. Simulation and Experimentation

The components and the framework can be used to simulate scenarios and provide a method of evaluating using a selection of metrics.

A. Metrics

In order to assess the performance of tracking and filtering algorithms, a variety of tracking metrics are included in the Stone Soup library. Track-to-truth metrics provide insight into how well a tracker was able to perform knowing the target truth data. These are often 'end-user-led', less formal figures of merit which enumerate information on the number of tracks produced (on both real and false measurements), how complete the overall track picture is, how many spurious or ambiguous tracks are included, the kinematic accuracy, track continuity, and so on. See for example [26],

For multi-target tracking scenarios, metrics which enumerate the total track uncertainty (in the information-theoretic sense) are provided. These are useful in sensor management objective functions, though they pay no heed to cardinality errors. Therefore, metrics which incorporate this consideration, like OSPA [27] or GOSPA [28] can be used. A high-performing multi-target tracker should not only aim to apply the correct measurement to the corresponding target but also reduce the errors that arise from missed or false targets. Once a track has been assigned to a target object, it is desirable to assess the quality of the filtered estimates. The Posterior Cramèr-Rao Lower Bound metric (e.g. [29]) provides a lower bound on the variance of the state estimate and can be used as such a benchmark.

B. Simulation

A key feature of Stone Soup's framework is the ability to model simple scenarios for assessing algorithm performance for a wide range of use cases. This has proven particularly useful for sensor management aspects in Stone Soup, enabling close interaction required algorithms and sensors, something not possible with recorded data sets.

A Stone Soup simulation is groundtruth led. A user has the choice to either read in groundtruth data using a `GroundTruthReader`, if they have their own data set, or to generate synthetic data using Stone Soup's simulators. These simulators can be used to rapidly create simple trajectories for objects on a small-scale.

Even though one can create detailed scenarios with relative ease using Stone Soup's base components, often it's desirable to model niche complex behaviours for your particular use case, something that the base components may not currently provide. These behaviours usually require a lot of hours and expertise to implement. Thankfully, there are solutions at hand, one of them being to utilize open-source software.

Specialised third party software can be used to act as a simulator. For example, SUMO ("Simulation of Urban Mobility") [30] is a popular open source simulation package that is tailored for simulating large traffic systems. Once a SUMO simulation is created, one can read in this data using a custom sub-class of a `GroundTruthReader`, namely SUMOGroundTruthReader into Stone Soup's `GroundTruthPath` data type.

Once groundtruth is in place, sensors can be simulated to produce sensor data. Stone Soup's public code base has a range of simple sensors. These sensors also serve as a handy blueprint for the user to implement their own, possibly more elaborate models. A simulated sensor in Stone Soup requires a measurement model, carrying information on how the sensor sees the world, and a measure method. This measure

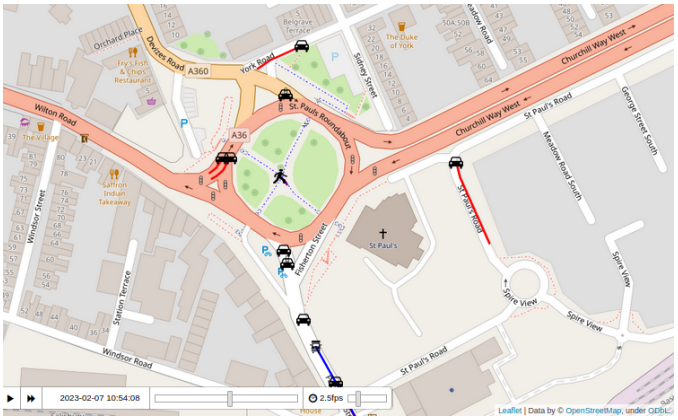

Fig. 8. A frame of a simple traffic simulation created with SUMO.

method processes the groundtruth objects and returns a set of detections.

It's important to be able to model the possibility of a false alarm when producing a simulation. Stone Soup's sensors have the ability to model false alarms and clutter, through the use of a `ClutterModel`. These models generate `Clutter` according to a user-specified distribution and clutter rate. `Clutter` detections are added to the set of detections returned by a sensor's measure method.

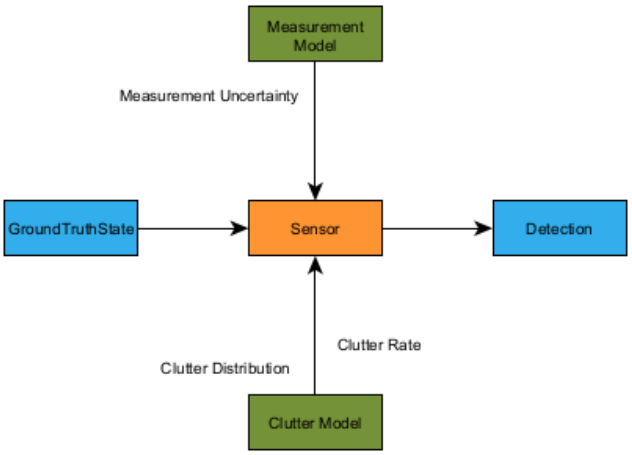

Fig. 9. The structure of a simple sensor in Stone Soup.

C. Experimentation

Processing multiple trackers with numerous different parameters can now be performed with the `RunManager`. This adds a command line interface (CLI) that allows multiple trackers, parameters or scenarios to be run from a single command. Stone Soup objects can be serialised into YAML. The run manager uses a YAML configuration file (using the `YAMLConfigWriter`) containing the following serialised components; tracker, ground truth and metric manager. Each scenario can be processed multiple times where the scenario includes random elements utilising Monte Carlo runs allowing the ability to execute a single tracker and producing average performance metrics over the scenario. Multiple Configuration runs can be run to execute a set of trackers, where one or more parameters or components are varied. Parameters to change for each tracker are defined in the JSON parameter file (created using the `JSONParameterWriter`). This can be extended to include Monte Carlo runs for each individual scenario. Given the combinatorics of multiple parameter changes and multiple Monte Carlo runs the ability to execute runs across multiple cores/processors to speed up execution is available.

Future work aims to implement a HPC compatible version of the run manager that can be run on AWS/Slurm HPCs. Additionally, to produce a run manager to do an optimisation run which is able to execute a single tracker, and vary one (or more) parameters to find the optimal value against a given metric.

D. Track Stitching

A new addition to the Stone Soup framework is the capability to 'stitch' tracks together using the `Stitcher` class. Given a set of disjoint tracks, `Stitcher` predicts the state forwards and backwards. Comparing the predicted state to the states of other track sections at the same point enables the association of track sections, ultimately outputting a set of complete tracks (where each track is made up of multiple sections). The stitcher algorithm works in N-dimensional space with higher dimensional track section data providing greater stitching results. A stitching metric is available which measures how well the stitcher algorithm is performing, by calculating the proportion of track sections that are stitched together correctly (providing this data is available).

E. Fusion Architectures

The capability to conduct simulations of different sensing architectures allows for testing and demonstration of their robustness, bandwidth requirements, effects of latency, and potential for data incest. A distinction is made between network architecture, which is how data is transmitted through the network, and information architecture, how it is accessed and processed. The information architecture is of course affected by the performance of the network architecture, and so for this reason the two counterparts for a given scenario can be contained together in `CombinedArchitecture`, providing this inter-dependence functionality. This section of Stone Soup makes use of the NetworkX [31] and GraphViz [32] packages, the latter of which is used in producing architectural plots.

IV. Applications & Communities

A. Applications

Stone Soup has been applied in many scenarios, from tracking vehicles in video, simulating radar coverage for air targets, orbital state estimation [6]. Application to radar sensors and tracking/classifying of multiple unmanned aircraft/drones has been a particularly active use case [33] [34] [35]. *MovingPandas* [36] is an example where Stone Soup has been used as a

dependency for another projects to add trajectory smoothing, applicable to multiple domains.

Being open source, there are also a multitude of other applications that Stone Soup is being applied to that the authors are not exposed to, but via the various §IV-B communities, have anecdotal evidence of application to: infra-red sensors, active/passive sonar, AIS[5] and radar fusion, satellite and inertial navigation systems fusion, fluorescence microscopy cell tracking.

Other applications include adding reinforcement learning (RL) into Stone Soup as a sensor management algorithm. In RL, an intelligent agent makes decisions from given observations, in an attempt to maximise a cumulative reward. In a sensor management scenario, the agent will take actions with a sensor, and receive rewards given by either a reward function provided by Stone Soup, or a handcrafted one for a given scenario. Work is being done on integrating RL into Stone Soup in the form of a `ReinforcementSensorManager`, which can be used to train an agent to interact with a user-defined environment, and develop a policy for choosing sensor actions from a given observation. The learning is accomplished with the Tensorflow Agents package [37], a framework for easily deploying and training RL algorithms.

B. Communities

There are a number of active communities advancing and developing Stone Soup for different applications. These cover academia, industry and government organisations engaging in common use cases or areas of interest, using Stone Soup as a framework to enable collaborative working. Public and open user community engagement takes place on forum-like platform *GitHub discussions*[6] and chat platform *Gitter*[7].

As part of the International Society of Information Fusion (ISIF), the Open-Source Tracking and Estimation Working Group (OSTEWG) aims to enable ISIF members to collaborate on the development of open source implementations for tracking, state estimation and fusion with Stone Soup. A NATO Research Task Group titled *Evaluation Framework for Multi-sensor Tracking and Fusion Algorithms* has been set up to build and evaluate tracking algorithms for defined military scenarios, applications and data sets.

Another key output of Stone Soup has also been the development of practical tutorials aimed at teaching the basic concepts of tracking and state estimation, with a focus on enabling easy application of algorithms to users who may lack background in the area of signal processing. These have been incorporated into training courses by various academic and industry organisations, and delivered by the core Stone Soup team at conferences such as ISIF Fusion conference [38], Institute of Electrical and Electronics Engineers (IEEE) International Instrumentation & Measurement Technology Conference (I2MTC) [39], University Defence Research Collaboration (UDRC) Summer School [40].

[5]Automatic Identification System (AIS) is a transceiver system for sharing identity, location, course, speed, etc between maritime platforms

[6]https://github.com/dstl/Stone-Soup/discussions

[7]https://gitter.im/dstl/Stone-Soup

V. Summary

This paper gives an overview of version 1.0 of Stone Soup, and has highlighted a subset of its important features. Stone Soup is already proving invaluable to multiple communities as a framework to apply and assess tracking and state estimation approaches. With this release it is the authors' hope that this milestone encourages additional adoption and contributions, especially within those communities looking for confidence in framework stability.

Development continues, focused on adding new features, algorithms, and maintaining backwards compatibility. Additional features of interest include, but are not limited to: Multiple Hypothesis Tracking (MHT), track before detect, extended object tracking, additional sensor modalities.

The authors wish to thank all those tracking users, professionals, enthusiasts, other community members, who've contributed to the project [1]. We welcome additional contributions, whether in implementations of algorithms, models, metrics, and documentation. As ever, additional peer review is valuable and sought after.

References

[1] S. Hiscocks, O. Harrald, J. Barr, N. Perree, L. Vladimirov, R. Green, E. Rogers, I. Dorrington, E. Hunter, J. Wright, O. Rosoman, H. Pritchett, J. Osborne, P. Carniglia, C. England, L. Flaherty, D. Kirkland, R. Davies, S. Naylor, and M. Campbell, "Stone Soup." [Online]. Available: https://doi.org/10.5281/zenodo.4663993

[2] P. Thomas, "Open letter on the "Stone Soup" tracking framework," in *Proceedings of the 10th Data Fusion and Target Tracking Conference, University of Liverpool, United Kingdom, 30 Apr. 2014.*, 2014.

[3] P. A. Thomas, J. Barr, B. Balaji, and K. White, "An open source framework for tracking and state estimation ('Stone Soup')," in *Signal Processing, Sensor/Information Fusion, and Target Recognition XXVI*, I. Kadar, Ed., vol. 10200, International Society for Optics and Photonics. SPIE, 2017, p. 1020008. [Online]. Available: https://doi.org/10.1117/12.2266249

[4] "Stone Soup alpha demonstration," 2018.

[5] D. Last, P. Thomas, S. Hiscocks, J. Barr, D. Kirkland, M. Rashid, S. B. Li, and L. Vladimirov, "Stone Soup: announcement of beta release of an open-source framework for tracking and state estimation," in *Signal Processing, Sensor/Information Fusion, and Target Recognition XXVIII*, I. Kadar, E. P. Blasch, and L. L. Grewe, Eds., vol. 11018, International Society for Optics and Photonics. SPIE, 2019, p. 1101807. [Online]. Available: https://doi.org/10.1117/12.2518514

[6] J. Barr, O. Harrald, S. Hiscocks, N. Perree, H. Pritchett, S. Vidal, J. Wright, P. Carniglia, E. Hunter, D. Kirkland, D. Raval, S. Zheng, A. Young, B. Balaji, S. Maskell, M. Hernandez, and L. Vladimirov, "Stone Soup open source framework for tracking and state estimation: enhancements and applications," in *Signal Processing, Sensor/Information Fusion, and Target Recognition XXXI*, I. Kadar, E. P. Blasch, and L. L. Grewe, Eds., vol. 12122, International Society

for Optics and Photonics. SPIE, 2022, p. 1212205. [Online]. Available: https://doi.org/10.1117/12.2618495

[7] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[8] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[9] G. Brandl, T. Shimizukawa, T. Komiya, T. Kampik, G. Varoquaux, J. Grobler, E. S. de Andrade, C. Holdgraf, A. Gramfort, M. Jas, J. Nothman, O. Grisel, N. Varoquaux, E. Gouillart, S. Hiscocks, T. Hoffmann, A. Lee, M. Luessi, S. Rehberg, alexis, G. Uberti, J. Vanderplas, T. A. Caswell, B. Sullivan, A. Y. Shih, A. Batula, P. Kunzmann, D. Stansby, D. Stańczak-Marikin, and J. Schueller, "The sphinx documentation builder." [Online]. Available: https://www.sphinx-doc.org/en/master/

[10] Ó. Nájera, E. Larson, L. Liu, L. Estève, G. Varoquaux, J. Grobler, E. S. de Andrade, C. Holdgraf, A. Gramfort, M. Jas, J. Nothman, O. Grisel, N. Varoquaux, E. Gouillart, S. Hiscocks, T. Hoffmann, A. Lee, M. Luessi, S. Rehberg, alexis, G. Uberti, J. Vanderplas, T. A. Caswell, B. Sullivan, A. Y. Shih, A. Batula, P. Kunzmann, D. Stansby, D. Stańczak-Marikin, and J. Schueller, "sphinx-gallery/sphinx-gallery," 2022. [Online]. Available: https://doi.org/10.5281/zenodo.3741780

[11] H. Krekel, B. Oliveira, R. Pfannschmidt, F. Bruynooghe, B. Laugher, and F. Bruhin, "pytest," 2004. [Online]. Available: https://github.com/pytest-dev/pytest

[12] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME – Journal of Basic Engineering*, vol. 82, 1960.

[13] P. S. Maybeck, *Stochastic Models, Estimation, and Control*, ser. ISSN. Elsevier Science, 1982. [Online]. Available: https://books.google.co.uk/books?id=L_YVMUJKNQUC

[14] Y.-S. Kim and K.-S. Hong, "Decentralized information filter in federated form," in *SICE 2003 Annual Conference (IEEE Cat. No.03TH8734)*, vol. 2, 2003, pp. 2176–2181 Vol.2.

[15] H. Tanizaki and R. S. Mariano, "Nonlinear filters based on taylor series expansions," *Communications in Statistics - Theory and Methods*, vol. 25, no. 6, pp. 1261–1282, 1996. [Online]. Available: http://dx.doi.org/10.1080/03610929608831763

[16] S. Julier, J. Uhlmann, and H. F. Durrant-Whyte, "A new method for the nonlinear transformation of means and covariances in filters and estimators," *IEEE Transactions on Automatic Control*, vol. 45, no. 3, pp. 477–482, 2000.

[17] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, "Novel approach to nonlinear/non-gaussian bayesian state estimation," *IEE Proceedings F - Radar and Signal Processing*, vol. 140, no. 2, pp. 107–113, April 1993.

[18] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002.

[19] F. Daum, J. Huang, and A. Noushin, "Exact particle flow for nonlinear filters," in *Signal Processing, Sensor Fusion, and Target Recognition XIX*, I. Kadar, Ed., vol. 7697, International Society for Optics and Photonics. SPIE, 2010, p. 769704. [Online]. Available: https://doi.org/10.1117/12.839590

[20] ——, "Generalized Gromov method for stochastic particle flow filters," in *Signal Processing, Sensor/Information Fusion, and Target Recognition XXVI*, I. Kadar, Ed., vol. 10200, International Society for Optics and Photonics. SPIE, 2017, p. 102000I. [Online]. Available: https://doi.org/10.1117/12.2248723

[21] D. E. Clark, K. Panta, and B.-N. Vo, "The GM-PHD filter multiple target tracker," in *2006 9th International Conference on Information Fusion*, 2006, pp. 1–8.

[22] Y. Tao, D. Papadias, and J. Sun, "The TPR*-Tree: An optimized spatio-temporal access method for predictive queries," in *Proceedings of 29th International Conference on Very Large Data Bases, VLDB 2003, Berlin, Germany, September 9-12, 2003*, J. C. Freytag, P. C. Lockemann, S. Abiteboul, M. J. Carey, P. G. Selinger, and A. Heuer, Eds. Morgan Kaufmann, 2003, pp. 790–801. [Online]. Available: http://www.vldb.org/conf/2003/papers/S24P01.pdf

[23] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, p. 509–517, sep 1975. [Online]. Available: https://doi.org/10.1145/361002.361007

[24] A. Hero, D. Castañón, D. Cochran, and K. Kastella, *Foundations and Applications of Sensor Management*, ser. Signals and Communication Technology. Springer US, 2007. [Online]. Available: https://books.google.co.uk/books?id=ffvPAAAACAAJ

[25] V. Krishnamurthy, *Partially Observed Markov Decision Processes: From Filtering to Controlled Sensing*. Cambridge University Press, 2016.

[26] S. Karoly, J. Wilson, H. Dutchyshyn, and J. Maluda, "Single integrated air picture (SIAP) attributes version 2.0," *Technical Report*, p. 53, 08 2003. [Online]. Available: https://apps.dtic.mil/sti/citations/ADA420755

[27] D. Schuhmacher, B.-T. Vo, and B.-N. Vo, "A consistent metric for performance evaluation of multi-object filters," *IEEE Transactions on Signal Processing*, vol. 56, no. 8, pp. 3447–3457, 2008.

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

[29] M. Hernandez, A. Farina, and B. Ristic, "PCRLB for tracking in cluttered environments: measurement sequence conditioning approach," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 42, no. 2, pp. 680–704, 2006.

[30] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. [Online]. Available: https://elib.dlr.de/124092/

[31] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.

[32] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, "Graphviz— open source graph drawing tools," in *Graph Drawing*, P. Mutzel, M. Jünger, and S. Leipert, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 483–484.

[33] P. Sévigny, D. Kirkland, X. Li, and B. Balaji, "Unmanned aircraft (UA) telemetry data for track modelling and classification," in *STO Meeting Proceedings*, 2021.

[34] C. Coman, A. Ndoni, S. Hiscocks, A. Lalas, R. Saygili, and P. Medentzidou, "ICMCIS'21 C-UAS data challenge," in *Proc. International Conference on Military Communications and Information Systems 2021*, 2021.

[35] P. Carniglia, B. Balaji, and A. Damini, "Investigation of sensor bias and signal quality on target tracking with multiple radars," in *2022 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2022, pp. 1–6.

[36] A. Graser, R. Bell, G. P. Boiko, A. Parnell, L. Vladimirov, G. S. Theodoropoulos, G. Richter, J. B. Elcinto, J. L. C. Rodríguez, R. Lovelace, A. Pulver, B. Larsen, Filipe, J. Gaboardi, L. Lengrand, L. Delucchi, M. Fleischmann, M. Kuhn, R. Miguel, S. Jozefowicz, S. Menegon, Susan, git it, radical squared, rlukevie, stevemarin, and tsuga, "anitagraser/movingpandas: v0.15," Jan. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7581454

[37] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartók, J. Berent, C. Harris, V. Vanhoucke, and E. Brevdo, "TF-Agents: A library for reinforcement learning in tensorflow," https://github.com/tensorflow/agents, 2018, [Online; accessed 25-June-2019]. [Online]. Available: https://github.com/tensorflow/agents

[38] "Tutorial - Stone Soup: an open-source tracking and state estimation framework; principles, use and applications," 2022. [Online]. Available: https://doi.org/10.23919/fusion49751.2022

[39] "Tutorial - Stone Soup: an open-source tracking and state estimation framework; principles, use and applications," 2022. [Online]. Available: https://doi.org/10.1109/i2mtc48687.2022

[40] "UDRC-EURASIP Summer School 2022." [Online]. Available: https://udrc.eng.ed.ac.uk/udrc-eurasip-summer-school-2022