# Modular Tracking Framework: A Unified Approach to Registration based Tracking

Abhineet Singh, Martin Jagersand
Department of Computing Science
University of Alberta, Edmonton, Canada
asingh1@ualberta.ca, jag@cs.ualberta.ca

***Abstract*— This paper presents a modular, extensible and highly efficient open source framework for registration based tracking targeted at robotics applications. It is implemented entirely in C++ and is designed from the ground up to easily integrate with systems that support any of several major vision and robotics libraries including OpenCV, ROS, ViSP and Eigen. It is also faster and more precise than other existing systems.**

**To establish the theoretical basis for its design, a new way to conceptualize registration based trackers is introduced that decomposes them into three constituent sub modules - Search Method, Appearance Model and State Space Model. In the process, the seminal work by Baker & Matthews [17] is extended with several important advances since its publication.**

**In addition to being a practical solution for fast and high precision tracking, this system can also serve as a useful research tool by allowing existing and new methods for any of the sub modules to be studied better. When a new method is introduced for one of these, the breakdown can help to experimentally find the combination of methods for the others that is optimum for it. By extensive use of generic programming, the system makes it easy to plug in a new method for any of the sub modules so that it can not only be tested comprehensively with existing methods but also become immediately available for deployment in any project that uses the framework.**

## I. Introduction

Fast and high precision visual tracking is crucial to the success of several robotics applications like visual servoing and autonomous navigation. In recent years, online learning and detection based trackers have become popular [1], [2] due to their robustness to changes in the object's appearance which makes them better suited to long term tracking. Such trackers, however, are often unsuitable for robotics applications for two reasons. Firstly, they are too slow [3] to allow real time execution of tasks where multiple trackers have to be run simultaneously or tracking is only a small part of a larger system. Secondly, they are not precise enough [3] to provide the exact object pose with sub pixel alignment required for these tasks. As a result, registration based trackers (Sec. II-B) are more suitable for these applications as being several times faster and capable of estimating higher degree-of-freedom (DOF) transformations like affine and homography.

Though several major advances have been made in this domain since the original Lucas Kanade (LK) tracker was introduced almost thirty five years ago [4], efficient open source implementations of recent trackers are surprisingly difficult to find. In fact, the only such tracker offered by the popular OpenCV library [5], uses a pyramidal implementation of the original algorithm [6]. Similarly, the ROS library [7] currently does not have any package that implements a modern registration based tracker. The XVision system [8] did introduce a full tracking framework including a video pipeline. However, it implements several variants of the same algorithm [9] that only gives reasonable tracking performance with low DOF motion. In addition, it is not well documented and is quite difficult to install on modern systems due to many obsolete dependencies. Even the fairly recent MRPT library [10] includes only a version of the original LK tracker apart from a low DOF particle filter based tracker which is too imprecise and slow to be considered relevant for our target applications.

In the absence of good open source implementations of modern trackers, most robotics research groups either use these out dated trackers or implement their own custom trackers. These, in turn, are often not made publicly available or are tailored to suit very specific needs and so require significant reprogramming to be useful for an unrelated project. To address this need for a tracking library targeted specifically at robotics applications, we introduce Modular Tracking Framework (**MTF**)[1] - a generic system for registration based tracking that provides highly efficient implementations for a large subset of trackers introduced in literature to date and is designed to be easily extensible with additional methods.

MTF conceptualizes a registration based tracker as being composed of three semi independent sub modules - Search Method (**SM**), Appearance Model (**AM**) and State Space Model (**SSM**) - where the former is treated as a way to use the functionality in the other two - through a well defined interface - to solve the tracking problem. Such an approach can help to address another urgent need in this field - to unify the myriad of contributions made in the last three decades so they can be better understood. When a new registration based tracker is introduced in literature, it often contributes to only one or two of these sub modules while using existing methods for the rest. In such cases, this breakdown can provide a model within which the contributions of the new tracker can be clearly demarcated and thus studied better [3], [11]. By following this decomposition closely through

[1]available at http://webdocs.cs.ualberta.ca/~vis/mtf/

extensive use of generic programming, MTF provides a convenient interface to plug in a new method for any sub module and test it against existing methods for the other two. This will not only help to compare the new method against existing ones in a more comprehensive way but also make it immediately available to any project that uses MTF. To facilitate the latter, MTF provides a simple ROS interface[2] for seamless integration with robotics systems.

An existing system that is quite similar to MTF in functionality is the template tracker module of the Visual Servoing Platform (ViSP) library [12] that includes 4 SMs, 3 AMs and 6 SSMs though not all combinations work. MTF offers several advantages over ViSP. Firstly, SMs and AMs in ViSP are not implemented as independent modules, rather each combination of methods has its own class. This makes it difficult to add a new method for either of these sub modules and combine it with existing methods for the others. Secondly, MTF has several more AMs than ViSP as well as three stochastic SMs - NN [13], Particle Filter [14] and RANSAC [15]. Lastly, MTF is significantly faster than ViSP - both ICLK and FCLK (Sec. II-C) with SSD/Homography are nearly 30 times[3] faster. This is mainly because MTF uses the Eigen library - known to be one of the fastest [16] - for all mathematical computations.

To summarize, following are our main contributions:

- Present a fast tracking library for robotics applications that is also easy to extend due to its modular design.
- Provide a unifying formulation for registration based tracking to establish the theoretical basis for its design.
  - This can be seen as an extension of the framework reported in [17] with newer methods.

Rest of this paper is organized as follows: Section II introduces the mathematical basis for the design of MTF while section III describes the class structure of MTF along with specifications for important functions. Section IV presents several SMs as examples of using the functionality described in section III to implement the theory of section II. Finally, section V presents a couple of use cases for MTF before concluding in section VI with ideas for future extensions.

## II. Theoretical Background

### A. Notation

Let $I_t : \mathbb{R}^2 \mapsto \mathbb{R}$ refer to an image captured at time $t$ treated as a smooth function of real values using sub pixel interpolation [18] for non integral locations. The patch corresponding to the tracked object's location in $I_t$ is denoted by $\mathbf{I_t}(\mathbf{x_t}) \in \mathbb{R}^N$ where $\mathbf{x_t} = [\mathbf{x_{1t}}, ..., \mathbf{x_{Nt}}]$ with $\mathbf{x_{kt}} = [x_{kt}, y_{kt}]^T \in \mathbb{R}^2$ being the Cartesian coordinates of pixel $k$.

Further, $\mathbf{w}(\mathbf{x}, \mathbf{p_s}) : \mathbb{R}^2 \times \mathbb{R}^S \mapsto \mathbb{R}^2$ denotes a warping function of $S$ parameters that represents the set of allowable image motions of the tracked object by specifying the deformations that can be applied to $\mathbf{x_0}$ to align $\mathbf{I_t}(\mathbf{x_t}) = \mathbf{I_t}(\mathbf{w}(\mathbf{x_0}, \mathbf{p_{st}}))$ with $\mathbf{I_0}(\mathbf{x_0})$. Examples of $\mathbf{w}$ include homography, affine, similitude, isometry and translation [19].

[2]available at `https://gitlab.com/vis/mtf_bridge`

[3]2400 vs 87 fps and 615 vs 22 fps respectively on an Intel Core i5 system

Finally $f(\mathbf{I}^*, \mathbf{I^c}, \mathbf{p_a}) : \mathbb{R}^N \times \mathbb{R}^N \times \mathbb{R}^A \mapsto \mathbb{R}$ is a function of $A$ parameters that measures the similarity between two patches - the reference or template patch $\mathbf{I}^*$ and a candidate patch $\mathbf{I^c}$. Examples of $f$ with $A = 0$ include sum of squared differences (SSD) [9], sum of conditional variance (SCV) [20], normalized cross correlation (NCC) [21], mutual information (MI) [18] and cross cumulative residual entropy (CCRE) [22]. So far, the only examples with $A \neq 0$, to the best of our knowledge, are those with an illumination model (**ILM**) [23], [24] where $f$ is expressed as $f(\mathbf{I}^*, g(\mathbf{I^c}, \mathbf{p_a}))$ with $g : \mathbb{R}^N \times \mathbb{R}^A \mapsto \mathbb{R}^N$ accounting for differences in lighting conditions under which $I_0$ and $I_t$ were captured.

### B. Registration based tracking

Using this notation, registration based tracking can be formulated (Eq 1) as a search problem where the goal is to find the optimal parameters $\mathbf{p_t} = [\mathbf{p_{st}}, \mathbf{p_{at}}] \in \mathbb{R}^{S+A}$ that maximize the similarity, measured by $f$, between the target patch $\mathbf{I}^* = \mathbf{I_0}(\mathbf{x_0})$ and the warped image patch $\mathbf{I^c} = \mathbf{I_t}(\mathbf{w}(\mathbf{x_0}, \mathbf{p_t}))$, that is,

$$\mathbf{p_t} = \underset{\mathbf{p_s}, \mathbf{p_a}}{\operatorname{argmax}} \ f(\mathbf{I_0}(\mathbf{x_0}), \mathbf{I_t}(\mathbf{w}(\mathbf{x_0}, \mathbf{p_s})), \mathbf{p_a}) \qquad (1)$$

As has been observed before [19], [22], this formulation gives rise to an intuitive way to decompose the tracking task into three modules - the similarity metric $f$, the warping function $\mathbf{w}$ and the optimization approach. These can be designed to be semi independent in the sense that any given optimizer can be applied unchanged to several combinations of methods for the other two modules which in turn interact only through a well defined and consistent interface. In this work, we refer to these respectively as AM, SSM and SM.

### C. Gradient Descent and the Chain Rule

Though several types of SMs have been reported in literature, gradient descent based methods [4] are most widely used due to their speed and simplicity. As mentioned in [17], the LK tracker can be formulated in four different ways depending on which image is searched for the warped patch - $I_0$ or $I_t$ - and how $\mathbf{p_s}$ is updated in each iteration - additive or compositional. The four resultant formulations are thus called Forward Additive (**FALK**) [4], Inverse Additive (**IALK**) [9], Forward Compositional (**FCLK**) [25] and Inverse Compositional (**ICLK**) [26]. There is also a more recent approach called Efficient Second order Minimization (**ESM**) [27] that tries to make the best of both ICLK and FCLK by using information from both $I_0$ and $I_t$.

What all these methods have in common is that they solve Eq 1 by estimating an incremental update $\Delta\mathbf{p_t}$ to the optimal parameters $\mathbf{p_{t-1}}$ at time $t-1$ using some variant of the Newton method as:

$$\Delta\mathbf{p_t} = -\hat{\mathbf{H}}^{-1}\hat{\mathbf{J}}^T \qquad (2)$$

where $\hat{\mathbf{J}}$ and $\hat{\mathbf{H}}$ respectively are estimates for the Jacobian $\mathbf{J} = \partial f / \partial \mathbf{p}$ and the Hessian $\mathbf{H} = \partial^2 f / \partial \mathbf{p}^2$ of $f$ w.r.t. $\mathbf{p}$. For any formulation that seeks to decompose this class of trackers (among others) in the aforementioned manner, the chain rule for first and second order derivatives is indispensable and the resultant decompositions for $\mathbf{J}$ and $\mathbf{H}$ are given by Eqs. 3 and 4 respectively, assuming $A = 0$ (or $\mathbf{p} = \mathbf{p_s}$) for simplicity.

$$\mathbf{J} = \frac{\partial f(\mathbf{I}(\mathbf{w}(\mathbf{p})))}{\partial \mathbf{p}} = \frac{\partial f}{\partial \mathbf{I}} \nabla \mathbf{I} \frac{\partial \mathbf{w}}{\partial \mathbf{p}} \tag{3}$$

$$\mathbf{H} = \frac{\partial \mathbf{I}}{\partial \mathbf{p}}^T \frac{\partial^2 f}{\partial \mathbf{I}^2} \frac{\partial \mathbf{I}}{\partial \mathbf{p}} + \frac{\partial f}{\partial \mathbf{I}} \frac{\partial^2 \mathbf{I}}{\partial \mathbf{p}^2} \tag{4}$$

with $\dfrac{\partial \mathbf{I}}{\partial \mathbf{p}} = \nabla \mathbf{I} \dfrac{\partial \mathbf{w}}{\partial \mathbf{p}}$ and $\dfrac{\partial^2 \mathbf{I}}{\partial \mathbf{p}^2} = \dfrac{\partial \mathbf{w}}{\partial \mathbf{p}}^T \nabla^2 \mathbf{I} \dfrac{\partial \mathbf{w}}{\partial \mathbf{p}} + \nabla \mathbf{I} \dfrac{\partial^2 \mathbf{w}}{\partial \mathbf{p}^2}$. It follows that the AM computes terms involving $\mathbf{I}$ and $f$ ($\nabla \mathbf{I}$, $\nabla^2 \mathbf{I}$, $\partial f / \partial \mathbf{I}$ and $\partial^2 f / \partial \mathbf{I}^2$ ) while the SSM computes those with $\mathbf{w}$ ($\partial \mathbf{w} / \partial \mathbf{p}$, $\partial^2 \mathbf{w} / \partial \mathbf{p}^2$). Further, these generic expressions do not give the whole scope of the decompositions since the exact forms of $\hat{\mathbf{J}}$ and $\hat{\mathbf{H}}$ as well as the way these are split vary for different variants of LK. The reader is referred to [17] for more details though formulations relevant to the functions in MTF (Tables I and II), including several extensions to [17], are also presented in the appendix.

### D. Stochastic Search

A limitation of gradient descent type SMs is that they are prone to getting stuck in local maxima of $f$ especially when the object's appearance changes due to factors like occlusions, motion blur or illumination variations. An alternative approach to avoid this problem is to use stochastic search so as to cover a larger portion of the search space of $\mathbf{p}$. There are currently three main categories of such methods in our framework - particle filters (**PF**) [14], nearest neighbor (**NN**) [13] and RANSAC [15].

These SMs work by generating a set of random samples for $\mathbf{p}$ and evaluating the goodness of each by some measure of similarity with the template. Their performance thus depends mostly on the number and quality of stochastic samples used. While the former is limited only by the available computational resources, the latter is a bit harder to guarantee for a general SSM/AM. For methods that draw samples from a Gaussian distribution, the quality thereof is determined by the covariance matrix used and, to the best of our knowledge, no widely accepted method exists to estimate it in the general case. Most works either use heuristics or perform extensive hand tuning to get acceptable results [14].

Given this, a reasonable way to decompose these methods to fit our framework is to delegate the responsibility of generating the set of samples and estimating its mean entirely to the SSM and AM while letting the latter evaluate the suitability of each sample by providing the likelihood of the corresponding patch. Since the definition of what constitutes a good sample and how the mean of a sample set is to be evaluated depends on the SSM/AM, as do any heuristics for generating these samples (like the variance for each component of $\mathbf{p}$), such a decomposition ensures both theoretical validity and good performance in practice.

## III. System Design

As shown in the class diagram in Fig. 1, MTF closely follows the decomposition described in the previous section and has three abstract base classes corresponding to the three sub modules - `SearchMethod`, `AppearanceModel` and `StateSpaceModel`. [4] Of these, only `SM` is a generic class that is templated on specializations of the other two classes. A concrete tracker, defined as a particular combination of the three sub modules, thus corresponds to a subclass of `SM` that has been instantiated with subclasses of `AM` and `SSM`.

It may be noted that `SM` itself derives from a non generic base class called `TrackerBase` for convenient creation and interfacing of objects corresponding to heterogeneous trackers, including those external to MTF, so that they can be run simultaneously and their results combined to create a tracker more robust than any of its components. Allowing a diverse variety of trackers to integrate seamlessly is one of the core design objectives of MTF and this is emphasized by having such trackers derive from a separate base class called `CompositeBase`. Since individual registration based trackers are well known to be prone to failures and more than three decades of research has failed to make significant improvements in this regard, the composite approach seems to be one of the more promising ones [15]. MTF has thus been designed to facilitate work in this direction.

A particular SM in our formulation is defined only by its objective - to find the $\mathbf{p}$ that maximizes the similarity measure defined by the AM. Thus, different implementations of `SM` can cover a potentially wide range of methods that have little in common. As a result, `SM` is the least specific of these classes and only provides functions to initialize, update and reset the tracker along with accessors to obtain its current state. In fact, an SM is regarded in this framework simply as one way to *use* the methods provided by the other two sub modules to accomplish the above objective with the idea being to abstract out as much computation from the SM to the AM/SSM as possible so as to make for a general purpose tracker. Therefore, this section describes only `AM` and `SSM` in detail while some of the SMs currently available in MTF are presented in the next section as examples of using the functionality described here to carry out the search in different ways.

Another consequence of this conceptual impreciseness of `SM` is that a specific SM may use only a small subset of the functionality provided by `AM`/`SSM`. For instance, gradient descent type SMs do not use the random sampling functions of `SSM` and conversely, stochastic SMs do not use the differential functions required by the former. This has two further implications. Firstly, the functionality set out in `AM` and `SSM` is not fixed but can change depending on the requirements of an SM, i.e. if a new SM is to be implemented that requires some functionality not present in the current specifications, the respective class can be extended to support it - as long as such an extension makes logical sense within the definition of that class. Secondly, it is not necessary for all combinations of AMs and SSMs to support all SMs. For instance a similarity measure does not need to be

[4] For brevity, these will be referred to as `SM`, `AM` and `SSM` respectively with the font serving to distinguish the *classes* from the corresponding *concepts*.

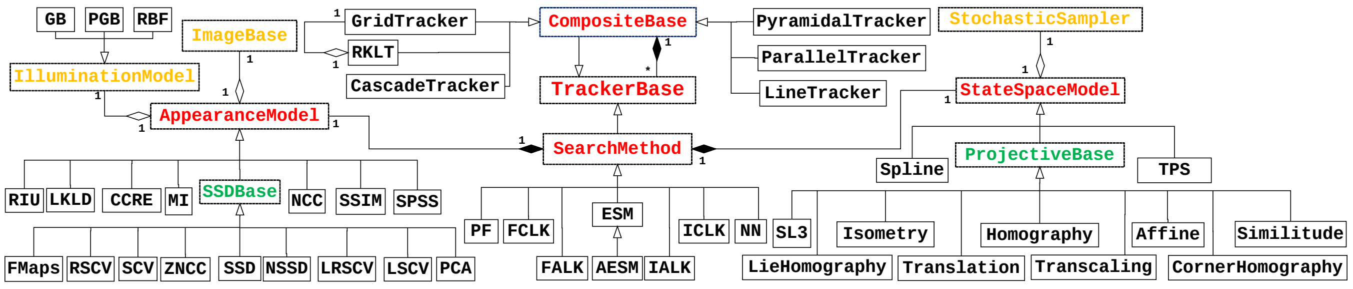

Fig. 1. MTF Class Diagram showing all models currently implemented. Pure and partially abstract classes are respectively shown in red and green while concrete classes are in black. Classes that are sub parts of AM and SSM are in yellow. Acronyms not defined in text: NSSD: Normalized SSD, ZNCC: Zero mean NCC, RSCV: Reversed SCV, LSCV: Localized SCV, LRSCV: Localized RSCV, PCA: Principal Component Analysis [28], SSIM: Structural Similarity[11], SPSS: Sum of Pixel wise SSIM[11], LKLD: Localized KL Divergence, RIU: Ratio Image Uniformity[29], GB: Gain and Bias[24], PGB: Piecewise GB[23], RBF: Radial Basis Function[23], TPS: Thin Plate Splines[30], Spline:[31], SL3:[14], AESM: Additive ESM, RKLT:[15]

differentiable to be a valid AM as long as it is understood that it cannot be used with SMs that require derivatives.

In the broadest sense, the division of functionality between AM and SSM described next can be understood as AM being responsible for everything to do with the image $I$, the sampled patch $\mathbf{I}(\mathbf{x})$ and the similarity $f$ computed using it while SSM handles the actual *points* $\mathbf{x}$ at which the patch is sampled along with the warping function $\mathbf{w}$ that defines it in terms of the state parameters $\mathbf{p}$.

## A. AppearanceModel

This class can be divided into three main parts with each defined as a set of variables dependent on $I_0$ and $I_t$ with a corresponding initialize and update function for each. The division is mainly conceptual and methods in different parts are free to interact with each other in practice. Table I presents a brief specification of some important methods in AM.

*1) Image Operations:* This part, abstracted into a separate class called ImageBase, handles all pixel level operations on the image $I$ like extracting the patch $\mathbf{I}(\mathbf{x})$ using sub pixel interpolation and computing its numerical gradient $\nabla\mathbf{I}$ and Hessian $\nabla^2\mathbf{I}$.

Though AM bears a composition or "has a" relationship with ImageBase, in practice the latter is actually implemented as a base class of the former to maintain simplicity of the interface and allow a specializing class to efficiently override functions in both classes. Moreover, having a separate class for pixel related operations means that AMs like SCV and ZNCC that differ from SSD only in using a modified version of $\mathbf{I_0}$ or $\mathbf{I_t}$ (thus deriving from SSDBase in Fig. 1), can implement the corresponding mapping entirely within the functions defined in ImageBase and be combined easily with other AMs besides SSD.

*2) Similarity Functions:* This is the core of AM and handles the computation of the similarity measure $f(\mathbf{I}^*, \mathbf{I^c}, \mathbf{p_a})$ and its derivatives $\partial f/\partial\mathbf{I}$ and $\partial^2 f/\partial\mathbf{I}^2$ w.r.t. both $\mathbf{I}^*$ and $\mathbf{I^c}$. It also provides interfacing functions to use inputs from SSM to compute the derivatives of $f$ w.r.t. SSM parameters using the chain rule. As a notational convention, all interfacing functions, including those in SSM, are prefixed with cmpt.

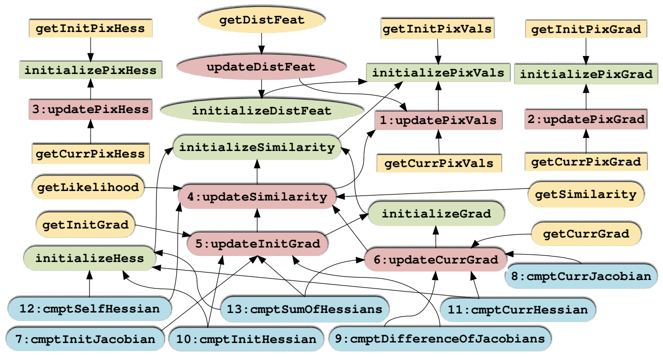

Fig. 2. Dependency relationships between various functions in AM: an arrow pointing from A to B means that A depends on B. Color of a function box denotes its type - green: initializing; red: updating; blue: interfacing and yellow: accessor function. Shape of a function box represents the part of AM it belongs to - rectangle: Image Operations; rounded rectangle: Similarity Functions; ellipse: Distance Feature. The numbers attached to some of the nodes refer Table I.

The functionality specific to $\mathbf{p_a}$ is abstracted into a separate class called IlluminationModel so it can be combined with any AM to add photometric parameters [23], [24] to it. This class provides functions to compute $g(\mathbf{I}, \mathbf{p_a})$ and its derivatives including $\partial g/\partial\mathbf{p_a}$, $\partial^2 g/\partial\mathbf{p_a}^2$, $\partial g/\partial\mathbf{I}$, $\partial^2 g/\partial\mathbf{I}^2$ and $\partial^2 g/\partial\mathbf{I}\partial\mathbf{p_a}$. These are called from within AM to compute the respective derivatives w.r.t. $f$ so that the concept of ILM is transparent to the SM. It should be noted that AM is designed to support $f$ with arbitrary $\mathbf{p_a}$ of which ILM is a special case. It also supports learning to update the object's appearance, as present, for instance, in PCA [28].

Since several of the functions in this part of AM involve common computations, there exist *transitive dependency* relationships between them as depicted in Fig. 2 to avoid repeating these computations when multiple quantities are needed by the SM. What this means is that a function lower down in the dependency hierarchy may delegate part of its computations to any function higher up in the hierarchy so that the latter must be called *before* calling the former if correct results are to be expected.

*3) Distance Feature:* This part is designed specifically to enable integration with the FLANN library [32] that is used by the NN based SM. It provides two main functions:

1) A feature transform $\mathbf{D}(\mathbf{I_1}) : \mathbb{R}^N \mapsto \mathbb{R}^K$ that maps the pixel values extracted from a patch $\mathbf{I_1}$ into a feature vector that contains the results of all computations in $f(\mathbf{I_1}, \mathbf{I_2})$ that depend only on $\mathbf{I_1}$.
2) A highly optimized distance functor $f_D(\mathbf{I_1^D}, \mathbf{I_2^D}) : \mathbb{R}^K \times \mathbb{R}^K \mapsto \mathbb{R}$ that computes a measure of the distance or dissimilarity between $\mathbf{I_1}$ and $\mathbf{I_2}$ (typically $-f(\mathbf{I_1}, \mathbf{I_2})$) given the distance features $\mathbf{I_1^D} = \mathbf{D}(\mathbf{I_1})$ and $\mathbf{I_2^D} = \mathbf{D}(\mathbf{I_2})$ as inputs.

The main idea behind the design of these two components is to place as much computational load as possible on $\mathbf{D}$ so that $f_D$ is as fast as possible with the premise that the former is called mostly during initialization when the sample dataset is to be built while the latter is called online to find the best matches for a candidate patch in the dataset.

TABLE I

SPECIFICATIONS FOR IMPORTANT METHODS IN AM. IDS IN FIRST COLUMN REFER FIG. 2

| ID | Inputs | Output/Variable updated |
|---|---|---|
| 1 | $\mathbf{x_t}$ | $\mathbf{I_t}(\mathbf{x_t})$ |
| 2 | $\mathbf{x_t}$ | $\nabla\mathbf{I_t}$ |
| 3 | $\mathbf{x_t}$ | $\nabla^2\mathbf{I_t}$ |
| 4 | None | $f(\mathbf{I_0}, \mathbf{I_t})$ |
| 5 | None | $\frac{\partial f(\mathbf{I_0}, \mathbf{I_t})}{\partial \mathbf{I_0}}$ |
| 6 | None | $\frac{\partial f(\mathbf{I_0}, \mathbf{I_t})}{\partial \mathbf{I_t}}$ |
| 7 | $\frac{\partial \mathbf{I_0}}{\partial \mathbf{p_s}}$ | $\frac{\partial f(\mathbf{I_0}(\mathbf{p}), \mathbf{I_t})}{\partial \mathbf{p}}$ (Eq. 8) |
| 8 | $\frac{\partial \mathbf{I_t}}{\partial \mathbf{p_s}}$ | $\frac{\partial f(\mathbf{I_0}, \mathbf{I_t}(\mathbf{p}))}{\partial \mathbf{p}}$ (Eq. 5, 6) |
| 9 | $\frac{\partial \mathbf{I_0}}{\partial \mathbf{p_s}}, \frac{\partial \mathbf{I_t}}{\partial \mathbf{p_s}}$ | $\frac{\partial f(\mathbf{I_0}, \mathbf{I_t}(\mathbf{p}))}{\partial \mathbf{p}} - \frac{\partial f(\mathbf{I_0}(\mathbf{p}), \mathbf{I_t})}{\partial \mathbf{p}}$ (Eq. 11) |
| 10[1] | $\frac{\partial \mathbf{I_0}}{\partial \mathbf{p_s}}, \frac{\partial^2 \mathbf{I_0}}{\partial \mathbf{p_s}^2}$ | $\frac{\partial^2 f(\mathbf{I_0}(\mathbf{p}), \mathbf{I_t})}{\partial \mathbf{p}^2}$ |
| 11 | $\frac{\partial \mathbf{I_t}}{\partial \mathbf{p_s}}, \frac{\partial^2 \mathbf{I_t}}{\partial \mathbf{p_s}^2}$ | $\frac{\partial^2 f(\mathbf{I_0}, \mathbf{I_t}(\mathbf{p}))}{\partial \mathbf{p}^2}$ |
| 12 | $\frac{\partial \mathbf{I_t}}{\partial \mathbf{p_s}}, \frac{\partial^2 \mathbf{I_t}}{\partial \mathbf{p_s}^2}$ | $\frac{\partial^2 f(\mathbf{I_t}, \mathbf{I_t}(\mathbf{p}))}{\partial \mathbf{p}^2}$ (Eq. 13, 14) |
| 13 | $\frac{\partial \mathbf{I_0}}{\partial \mathbf{p_s}}, \frac{\partial^2 \mathbf{I_0}}{\partial \mathbf{p_s}^2}, \frac{\partial \mathbf{I_t}}{\partial \mathbf{p_s}}, \frac{\partial^2 \mathbf{I_t}}{\partial \mathbf{p_s}^2}$ | $\frac{\partial^2 f(\mathbf{I_0}(\mathbf{p}), \mathbf{I_t})}{\partial \mathbf{p}^2} + \frac{\partial^2 f(\mathbf{I_0}, \mathbf{I_t}(\mathbf{p}))}{\partial \mathbf{p}^2}$ |

[1] Functions 10-13 have overloaded variants that omit the second term in Eq. 4, as in Eq. 12, and so do not require $\frac{\partial^2 \mathbf{I}}{\partial \mathbf{p_s}^2}$ as input

## B. `StateSpaceModel`

This class has a simpler internal state than AM and can be described by only three main variables at any time $t$ - sampled grid points $\mathbf{x_t}$, corresponding corners $\mathbf{x_t^c}$ and state parameters $\mathbf{p_{st}}$. It may be noted (Fig. 1) that, though SSM is designed to support any arbitrary $\mathbf{w}$, most SSMs currently implemented are subsets of the planar projective transform and so derive from `ProjectiveBase` that abstracts out the functionality common to these.

Functions in SSM can be divided into two categories:

*1) Warping Functions:* This is the core of SSM and provides a function $\mathbf{w}$ to transform a regularly spaced grid of points $\mathbf{x_0}$ representing the target patch into a warped patch $\mathbf{x_t} = \mathbf{w}(\mathbf{x_0}, \mathbf{p_{st}})$ that captures the tracked object's motion in image space. It also allows for the compositional inverse of $\mathbf{w}$ to be computed (`invertState`) to support inverse SMs. Further, there are functions to compute the derivatives of $\mathbf{w}$ w.r.t. both $\mathbf{x}$ and $\mathbf{p_s}$ but, unlike AM, SSM does not store these as state variables, rather their computation is implicit in the interfacing functions that compute $\partial \mathbf{I}/\partial \mathbf{p_s}$ and $\partial^2 \mathbf{I}/\partial \mathbf{p_s}^2$ using chain rule. This design decision was made for reasons of efficiency since $\partial \mathbf{w}/\partial \mathbf{p_s}$ and $\partial \mathbf{w}/\partial \mathbf{x}$ are large and often very sparse tensors and computing these separately not only wastes a lot of memory but is also very computationally inefficient.

Finally, there are four ways to update the internal state: incrementally using additive (`additiveUpdate`) or compositional (`compositionalUpdate`) formulations, or outright by providing either the state vector (`setState`) or the corresponding corners (`setCorners`) that define the current location of the patch. There are no complex dependencies in SSM - the correct performance of interfacing functions and accessors depends only on one of the update functions being called every iteration. Table II lists the functionality of some important methods in this part.

TABLE II

SPECIFICATIONS FOR IMPORTANT METHODS IN SSM.

| Function | Inputs | Output/Result |
|---|---|---|
| `compositionalUpdate` | $\Delta\mathbf{p_s}$ | $\mathbf{p_{st}} = \mathbf{p_s}' \mid \mathbf{w}(\mathbf{x}, \mathbf{p_s}') = \mathbf{w}(\mathbf{w}(\mathbf{x}, \Delta\mathbf{p_s}), \mathbf{p_{st}})$ |
| `additiveUpdate` | $\Delta\mathbf{p_s}$ | $\mathbf{p_{st}} = \mathbf{p_{st}} + \Delta\mathbf{p_s}$ |
| `invertState` | $\mathbf{p_s}$ | $\mathbf{p_s'} \mid \mathbf{w}(\mathbf{w}(\mathbf{x}, \mathbf{p_s}), \mathbf{p_s'}) = \mathbf{x}$ |
| `cmptPixJacobian` | $\nabla\mathbf{I_t}$ | $\left.\frac{\partial \mathbf{I_t}}{\partial \mathbf{p_s}}\right\|_{\mathbf{p_s}=\mathbf{p_{st}}}$ (Eq. 5) |
| `cmptWarpedPixJacobian` | $\nabla\mathbf{I_t}$ | $\left.\frac{\partial \mathbf{I_t}(\mathbf{w})}{\partial \mathbf{p_s}}\right\|_{\mathbf{p_s}=\mathbf{p_{s0}}}$ (Eq. 6, 7) |
| `cmptApproxPixJacobian` | $\nabla\mathbf{I_0}$ | $\frac{\partial \mathbf{I_t}}{\partial \mathbf{p_{st}}}$ (approx) (Eq. 9, 10) |
| `cmptPixHessian` | $\nabla\mathbf{I_t}, \nabla^2\mathbf{I_t}$ | $\left.\frac{\partial^2 \mathbf{I_t}}{\partial \mathbf{p_s}^2}\right\|_{\mathbf{p_s}=\mathbf{p_{st}}}$ (Eq. 15) |
| `cmptWarpedPixHessian` | $\nabla\mathbf{I_t}, \nabla^2\mathbf{I_t}$ | $\left.\frac{\partial^2 \mathbf{I_t}(\mathbf{w})}{\partial \mathbf{p_s}^2}\right\|_{\mathbf{p_s}=\mathbf{p_{s0}}}$ (Eq. 16, 17) |
| `cmptApproxPixHessian` | $\nabla\mathbf{I_0}, \nabla^2\mathbf{I_0}$ | $\frac{\partial^2 \mathbf{I_t}}{\partial \mathbf{p_{st}}^2}$ (approx) (Eq. 18) |

*2) Stochastic Sampler:* This part is provided to support stochastic SMs and offers following functionality to this end:

1) generate small random incremental updates to $\mathbf{p_s}$ (`generatePerturbation`) by drawing these from a zero mean normal distribution with either user provided or heuristically estimated (`estimateStateSigma`) variance.
2) generate stochastic state samples using the given state transition model - currently random walk (`additiveRandomWalk`) and first order auto regression (`additiveAutoRegression1`) are supported. There are also `compositional` variants.
3) estimate the mean of a set of samples of $\mathbf{p_s}$ (`estimateMeanOfSamples`)
4) estimate the best fit $\mathbf{p_s}$ from a set of original and warped point pairs (`estimateWarpFromPts`) using a robust method - currently RANSAC [15] and Least Median of Squares [33] are supported.

## IV. Examples of Search Methods

This section presents pseudo codes for several SMs to exemplify the usage of functions described in the previous section. Following are some points/conventions to be noted:

1) *am* and *ssm* respectively refer to instances of `AM` and `SSM` (or rather of specializations thereof)
2) only one iteration of the `update` function is shown and $A = 0$ has been assumed for simplicity
3) the procedure for using first order Hessian (Eq. 12) is demonstrated only for ICLK but should be obvious for other relevant methods too
4) different algorithms refer each other to save space and emphasize the parts they have in common
5) *flann* in NN is an instance of FLANN library [32] that can build an index from a set of samples and search it for a new candidate
6) variables used to store the results of computations are not described explicitly but their meaning should be clear from their names and context
7) only one state transition model is shown in PF though several are available (Sec. III-B.2)

**Algorithm 1** ICLK

```
 1: function initialize(corners)
 2:     ssm.initialize(corners)
 3:     am.initializePixVals(ssm.getPts())
 4:     am.initializePixGrad(ssm.getPts())
 5:     am.initializeSimilarity()
 6:     am.initializeGrad()
 7:     am.initializeHess()
 8:     pix_jacobian ← ssm.cmptWarpedPixJacobian(am.getInitPixGrad())
 9:     if use_second_order_hess then
10:         am.initializePixHess(ssm.getPts())
11:         pix_hessian ← ssm.cmptInitPixHessian(
                am.getInitPixHess(), am.getInitPixGrad())
12:         hessian←am.cmptSelfHessian(pix_jacobian, pix_hessian)
13:     else
14:         hessian←am.cmptSelfHessian(pix_jacobian)
15:     end if
16: end function
17: function update
18:     am.updatePixVals(ssm.getPts())
19:     am.updateSimilarity()
20:     am.updateInitGrad()
21:     pix_jacobian ← ssm.cmptWarpedPixJacobian(am.getInitPixGrad())
22:     jacobian←am.cmptInitJacobian(pix_jacobian)
23:     ssm_update ← $-$hessian.inverse()$*$jacobian
24:     inv_update←ssm.invertState(ssm_update)
25:     ssm.compositionalUpdate(inv_update)
26:     return ssm.getCorners()
27: end function
```

## V. Use Cases

This section presents two simple use cases for MTF:

1) Track an object in an image sequence (Algorithm 6)
2) Estimate the trajectory of a UAV in a large satellite image from images it took while flying over the region covered in the map (Algorithm 7).

It should be noted that MTF comes with an input module with wrappers for image capturing functions in OpenCV, ViSP and XVision as well as a preprocessing module with wrappers for OpenCV image filtering functions. Raw images acquired by the former need to be passed to the latter before they can be passed to the tracker. `GaussianSmoothing`

**Algorithm 2** FCLK

```
 1: function initialize(corners)
 2:     lines 2-7 of Algorithm 1
 3:     am.initializePixHess(ssm.getPts())
 4: end function
 5: function update
 6:     lines 18-19 of Algorithm 1
 7:     am.updateCurrGrad()
 8:     am.updatePixGrad(ssm.getPts())
 9:     am.updatePixHess(ssm.getPts())
10:     pix_jacobian ← ssm.cmptWarpedPixJacobian(am.getCurrPixGrad())
11:     pix_hessian ← ssm.cmptWarpedPixHessian(
            am.getCurrPixHess(), am.getCurrPixGrad())
12:     jacobian←am.cmptCurrJacobian(pix_jacobian)
13:     hessian ←am.cmptSelfHessian(pix_jacobian, pix_hessian)
14:     ssm_update ← $-$hessian.inverse()$*$jacobian
15:     ssm.compositionalUpdate(ssm_update)
16:     return ssm.getCorners()
17: end function
```

**Algorithm 3** ESM

```
 1: function initialize(corners)
 2:     lines 2-3 of Algorithm 2
 3:     init_pix_jacobian ← ssm.cmptWarpedPixJacobian(am.getInitPixGrad())
 4:     init_hessian←am.cmptSelfHessian(pix_jacobian, pix_hessian)
 5: end function
 6: function update
 7:     lines 6-11 of Algorithm 2
 8:     am.updateInitGrad()
 9:     jacobian←am.cmptDifferenceOfJacobians(init_pix_jacobian, pix_jacobian)
10:     curr_hessian ←am.cmptSelfHessian(pix_jacobian, pix_hessian)
11:     hessian ←init_hessian $+$ curr_hessian
12:     lines 14-16 of Algorithm 2
13: end function
```

**Algorithm 4** NN

```
 1: function initialize(corners)
 2:     lines 2-3 of Algorithm 1
 3:     state_sigma← ssm.estimateStateSigma()
 4:     ssm.initializeSampler(state_sigma)
 5:     am.initializeDistFeat()
 6:     for sample_id ← 1, no_of_samples do
 7:         ssm_updates.row(sample_id) ← ssm.generatePerturbation()
 8:         inv_update ←ssm.invertState(ssm_updates.row(sample_id))
 9:         ssm.compositionalUpdate(inv_update)
10:         am.updatePixVals(ssm.getPts())
11:         am.updateDistFeat()
12:         sample_dataset.row(sample_id) ← am.getDistFeat()
13:         ssm.compositionalUpdate(ssm_updates.row(sample_id))
14:     end for
15:     flann.buildIndex(sample_dataset)
16: end function
17: function update
18:     am.updatePixVals(ssm.getPts())
19:     am.updateDistFeat()
20:     nn_sample_id ← flann.searchIndex(am.getDistFeat())
21:     ssm.compositionalUpdate(ssm_updates.row(nn_sample_id))
22:     return ssm.getCorners()
23: end function
```

**Algorithm 5** PF

```
 1: function initialize(corners)
 2:     lines 2-4 of Algorithm 4
 3:     am.initializeSimilarity()
 4:     for particle_id ← 1, no_of_particles do
 5:         particles[particle_id].state ← ssm.getState()
 6:         particles[particle_id].weight ← $1/$no_of_particles
 7:     end for
 8: end function
 9: function update
10:     for particle_id ← 1, no_of_particles do
11:         particles[particle_id].state ← ssm.compositionalRandomWalk(
12:             particles[particle_id].state)
13:         ssm.setState(particles[particle_id].state)
14:         am.updatePixVals(ssm.getPts())
15:         am.updateSimilarity()
16:         particles[particle_id].weight ←am.getLikelihood()
17:     end for
18:     normalize weights and resample the particles
19:     mean_state ← ssm.estimateMeanOfSamples(particles);
20:     ssm.setState(mean_state)
21:     return ssm.getCorners()
22: end function
```

**Algorithm 6** Track an Object

```
using namespace mtf;
FCLK<SSD, Homography> tracker;
GaussianSmoothing pre_proc(input.getFrame(), tracker.inputType());
tracker.initialize(pre_proc.getFrame(), init_location);
while input.update() do
    pre_proc.update(input.getFrame());
    tracker.update(pre_proc.getFrame());
    new_location ← tracker.getRegion();
end while
```

**Algorithm 7** Estimate UAV Trajectory in Satellite Image

```
ESM<MI, Similitude> uav_tracker;
uav_img_corners←getFrameCorners(input.getFrame());
GaussianSmoothing satellite_pre_proc(satellite_img, uav_tracker.inputType());
GaussianSmoothing uav_pre_proc(input.getFrame(), uav_tracker.inputType());
uav_tracker.initialize(satellite_pre_proc.getFrame(), init_uav_location);
curr_uav_location←uav_tracker.getRegion();
while input.update() do
    uav_pre_proc.update(input.getFrame());
    uav_tracker.initialize(uav_pre_proc.getFrame(), uav_img_corners);
    uav_tracker.setRegion(curr_uav_location);
    uav_tracker.update(satellite_pre_proc.getFrame());
    curr_uav_location←uav_tracker.getRegion();
end while
```

used in the algorithms is an example of the latter while the former is assumed to have been initialized with the appropriate input source. Also, though only a couple of combinations of SM, AM and SSM are shown here, these can be replaced by virtually any combination of methods. Finally, Algorithm 7 assumes that UAV images have approximately the same size as the corresponding region in the satellite image. Due to space limitations, experimental tracking results are not included here - these are available in [3] and [11].

## VI. Conclusions and Future Work

This paper presented a modular and extensible open source framework for registration based tracking that provides highly efficient C++ implementations for several well established trackers that will hopefully address practical tracking needs of the wider robotics community. It also formulated a novel method to decompose registration based trackers into sub modules and extended a previously published unifying formulation to account for more recent developments.

MTF is still a work in progress with several promising avenues of future extensions including novel composite SMs, incorporating better parameterization as well as deep learning in AMs, addition of non rigid SSMs and using motion learning to generate better stochastic samples for SSMs.

## Appendix: $\hat{\mathbf{J}}$ and $\hat{\mathbf{H}}$ for different variants of LK

### A. Jacobian

Denoting $\mathbf{w}(\mathbf{x},\mathbf{p_s})$ with $\mathbf{w}(\mathbf{p})$ for conciseness ($A=0$ and $\mathbf{x}$ is constant in this context) and letting $\mathbf{\hat{p}_t}$ denote an estimate of $\mathbf{p_t}$ to which an incremental update is sought, the formulations for $\hat{\mathbf{J}}$ used by FALK and FCLK are:

$$\hat{\mathbf{J}}_{fa} = \left.\frac{\partial f}{\partial \mathbf{I^c}}\right|_{\mathbf{I^c}=\mathbf{I_t}(\mathbf{w}(\mathbf{\hat{p}_t}))} \left.\nabla \mathbf{I_t}\right|_{\mathbf{x}=\mathbf{w}(\mathbf{\hat{p}_t})} \left.\frac{\partial \mathbf{w}}{\partial \mathbf{p}}\right|_{\mathbf{p}=\mathbf{\hat{p}_t}} \tag{5}$$

$$\hat{\mathbf{J}}_{fc} = \left.\frac{\partial f}{\partial \mathbf{I^c}}\right|_{\mathbf{I^c}=\mathbf{I_t}(\mathbf{w}(\mathbf{\hat{p}_t}))} \left.\nabla \mathbf{I_t}(\mathbf{w})\right|_{\mathbf{x}=\mathbf{x_0}} \left.\frac{\partial \mathbf{w}}{\partial \mathbf{p}}\right|_{\mathbf{p}=\mathbf{p_0}} \tag{6}$$

where $\nabla \mathbf{I_t}(\mathbf{w})$ in Eq. 6 refers to the gradient of $I_t$ warped using $\mathbf{\hat{p}_t}$, i.e. $I_t$ is first warped back to the coordinate frame of $I_0$ using $\mathbf{w}(\mathbf{\hat{p}_t})$ to obtain $I_t(\mathbf{w})$ whose gradient is then computed at $\mathbf{x}=\mathbf{x_0}$. It can be further expanded [17] as:

$$\left.\nabla \mathbf{I_t}(\mathbf{w})\right|_{\mathbf{x}=\mathbf{x_0}} = \left.\nabla \mathbf{I_t}\right|_{\mathbf{x}=\mathbf{w}(\mathbf{\hat{p}_t})} \left.\frac{\partial \mathbf{w}}{\partial \mathbf{x}}\right|_{\mathbf{p}=\mathbf{\hat{p}_t}} \tag{7}$$

Since $\nabla \mathbf{I_t}$ is usually the most computationally intensive part of $\mathbf{J}_{fc}$ and $\mathbf{J}_{fa}$, the so called inverse methods approximate this with the gradient of $\nabla \mathbf{I_0}$ for efficiency as this only needs to be computed once. The specific expressions for these methods are:

$$\hat{\mathbf{J}}_{ic} = \left.\frac{\partial f}{\partial \mathbf{I^*}}\right|_{\mathbf{I^*}=\mathbf{I_0}(\mathbf{x_0})} \left.\nabla \mathbf{I_0}\right|_{\mathbf{x}=\mathbf{x_0}} \left.\frac{\partial \mathbf{w}}{\partial \mathbf{p}}\right|_{\mathbf{p}=\mathbf{p_0}} \tag{8}$$

$$\hat{\mathbf{J}}_{ia} = \left.\frac{\partial f}{\partial \mathbf{I^c}}\right|_{\mathbf{I^c}=\mathbf{I_t}(\mathbf{w}(\mathbf{\hat{p}_t}))} \left.\nabla \mathbf{I_0}\right|_{\mathbf{x}=\mathbf{x_0}} \left.\frac{\partial \mathbf{w}}{\partial \mathbf{x}}^{-1}\right|_{\mathbf{p}=\mathbf{\hat{p}_t}} \left.\frac{\partial \mathbf{w}}{\partial \mathbf{p}}\right|_{\mathbf{p}=\mathbf{\hat{p}_t}} \tag{9}$$

where the middle two terms in Eq. 9 are derived from Eqs. 5 and 7 by assuming [9] that $\mathbf{w}(\mathbf{\hat{p}_t})$ perfectly aligns $I_t$ with $I_0$, i.e. $I_t(\mathbf{w}) = I_0$ so that

$$\nabla \mathbf{I_t}(\mathbf{w}) = \nabla \mathbf{I_0} \tag{10}$$

In its original paper [27], ESM was formulated as using the mean of the pixel gradients $\nabla \mathbf{I_0}$ and $\nabla \mathbf{I_t}(\mathbf{w})$ to compute $\mathbf{J}$ but, as this formulation is only applicable to SSD, we consider a generalized version [34], [21] that uses the *difference* between FCLK and ICLK Jacobians:

$$\hat{\mathbf{J}}_{esm} = \hat{\mathbf{J}}_{fc} - \hat{\mathbf{J}}_{ic} \tag{11}$$

### B. Hessian

For clarity and brevity, evaluation points for the various terms have been omitted in the equations that follow as being obvious from analogy with the previous section.

It is generally assumed [17], [27] that the second term of Eq. 4 is too costly to compute and too small near convergence to matter and so is omitted to give the so called Gauss Newton Hessian

$$\mathbf{H}_{gn} = \frac{\partial \mathbf{I}}{\partial \mathbf{p}}^T \frac{\partial^2 f}{\partial \mathbf{I}^2} \frac{\partial \mathbf{I}}{\partial \mathbf{p}} \tag{12}$$

Though $\mathbf{H}_{gn}$ works very well for SSD (and in fact even better than $\mathbf{H}$ [17], [18]), it is well known [18], [21] to *not* work well with other AMs like MI, CCRE and NCC for which an approximation to the Hessian *after convergence* has to be used by assuming perfect alignment or $\mathbf{I_t}(\mathbf{w}(\mathbf{\hat{p}_t})) = \mathbf{I_0}(\mathbf{x_0})$. We refer to the resultant approximation as the **Self Hessian** $\mathbf{H}_{self}$ and, as this substitution can be made by setting either $\mathbf{I^c} = \mathbf{I_0}(\mathbf{x_0})$ or $\mathbf{I^*} = \mathbf{I_t}(\mathbf{w}(\mathbf{\hat{p}_t}))$, we get two forms which are respectively deemed to be the Hessians for ICLK and FCLK:

$$\hat{\mathbf{H}}_{ic} = \mathbf{H}^*_{self} = \frac{\partial \mathbf{I_0}}{\partial \mathbf{p}}^T \frac{\partial^2 f(\mathbf{I_0},\mathbf{I_0})}{\partial \mathbf{I}^2} \frac{\partial \mathbf{I_0}}{\partial \mathbf{p}} + \frac{\partial f(\mathbf{I_0},\mathbf{I_0})}{\partial \mathbf{I}} \frac{\partial^2 \mathbf{I_0}}{\partial \mathbf{p}^2} \tag{13}$$

$$\hat{\mathbf{H}}_{fc} = \mathbf{H}^{\mathbf{c}}_{self} = \frac{\partial \mathbf{I_t}}{\partial \mathbf{p}}^T \frac{\partial^2 f(\mathbf{I_t},\mathbf{I_t})}{\partial \mathbf{I}^2} \frac{\partial \mathbf{I_t}}{\partial \mathbf{p}} + \frac{\partial f(\mathbf{I_t},\mathbf{I_t})}{\partial \mathbf{I}} \frac{\partial^2 \mathbf{I_t}}{\partial \mathbf{p}^2} \tag{14}$$

It is interesting to note that $\mathbf{H}_{gn}$ has the exact same form as $\mathbf{H}_{self}$ for SSD (since $\dfrac{\partial f_{ssd}(\mathbf{I_0},\mathbf{I_0})}{\partial \mathbf{I}} = \dfrac{\partial f_{ssd}(\mathbf{I_t},\mathbf{I_t})}{\partial \mathbf{I}} = \mathbf{0}$) so it seems that interpreting Eq. 12 as the first order approximation of Eq. 4, as in [17], [18], is incorrect and it should instead be seen as a special case of $\mathbf{H}_{self}$.

$\hat{\mathbf{H}}_{fa}$ differs from $\hat{\mathbf{H}}_{fc}$ only in the way $\dfrac{\partial^2 \mathbf{I_t}}{\partial \mathbf{p}^2}$ and $\dfrac{\partial \mathbf{I_t}}{\partial \mathbf{p}}$ are computed for the two as given in Eqs. 15 and 16 respectively.

$$\frac{\partial^2 \mathbf{I_t}}{\partial \mathbf{p}^2}(fa) = \frac{\partial \mathbf{w}}{\partial \mathbf{p}}^T \nabla^2 \mathbf{I_t} \frac{\partial \mathbf{w}}{\partial \mathbf{p}} + \nabla \mathbf{I_t} \frac{\partial^2 \mathbf{w}}{\partial \mathbf{p}^2} \tag{15}$$

$$\frac{\partial^2 \mathbf{I_t}}{\partial \mathbf{p}^2}(fc) = \frac{\partial \mathbf{w}}{\partial \mathbf{p}}^T \nabla^2 \mathbf{I_t}(\mathbf{w}) \frac{\partial \mathbf{w}}{\partial \mathbf{p}} + \nabla \mathbf{I_t}(\mathbf{w}) \frac{\partial^2 \mathbf{w}}{\partial \mathbf{p}^2} \tag{16}$$

where $\nabla^2 \mathbf{I_t}(\mathbf{w})$ can be expanded by differentiating Eq. 7 as:

$$\nabla^2 \mathbf{I_t}(\mathbf{w}) = \frac{\partial \mathbf{w}}{\partial \mathbf{x}}^T \nabla^2 \mathbf{I_t} \frac{\partial \mathbf{w}}{\partial \mathbf{x}} + \nabla \mathbf{I_t} \frac{\partial^2 \mathbf{w}}{\partial \mathbf{x}^2} \tag{17}$$

$\hat{\mathbf{H}}_{ia}$ is identical to $\hat{\mathbf{H}}_{fa}$ except that $\nabla \mathbf{I_0}$ and $\nabla^2 \mathbf{I_0}$ are used to approximate $\nabla \mathbf{I_t}$ and $\nabla^2 \mathbf{I_t}$. The expression for the former is in Eq. 9 while that for the latter can be derived by differentiating both sides of Eq. 7 after substituting Eq. 10:

$$\nabla^2 \mathbf{I_0} = \frac{\partial \mathbf{w}}{\partial \mathbf{x}}^T \nabla^2 \mathbf{I_t} \frac{\partial \mathbf{w}}{\partial \mathbf{x}} + \nabla \mathbf{I_t} \frac{\partial^2 \mathbf{w}}{\partial \mathbf{x}^2}$$

which gives:

$$\begin{aligned} \nabla^2 \mathbf{I_t}(ia) &= \left(\frac{\partial \mathbf{w}}{\partial \mathbf{x}}^{-1}\right)^T \left[\nabla^2 \mathbf{I_0} - \nabla \mathbf{I_t} \frac{\partial^2 \mathbf{w}}{\partial \mathbf{x}^2}\right] \frac{\partial \mathbf{w}}{\partial \mathbf{x}}^{-1} \\ &= \left(\frac{\partial \mathbf{w}}{\partial \mathbf{x}}^{-1}\right)^T \left[\nabla^2 \mathbf{I_0} - \left(\nabla \mathbf{I_0} \frac{\partial \mathbf{w}}{\partial \mathbf{x}}^{-1}\right) \frac{\partial^2 \mathbf{w}}{\partial \mathbf{x}^2}\right] \frac{\partial \mathbf{w}}{\partial \mathbf{x}}^{-1} \end{aligned} \tag{18}$$

where the second equality again follows from Eq. 7 and 10. Finally, the ESM Hessian corresponding to the Jacobian in Eq. 11 is the *sum* of FCLK and ICLK Hessians:

$$\hat{\mathbf{H}}_{esm} = \hat{\mathbf{H}}_{fc} + \hat{\mathbf{H}}_{ic} \tag{19}$$

## References

[1] Y. Wu, J. Lim, and M.-H. Yang, "Online Object Tracking: A Benchmark," in *CVPR*, June 2013, pp. 2411–2418.

[2] M. J. L. A. Kristan, Matej *et al.*, "The Visual Object Tracking VOT2015 Challenge Results," in *Proceedings of the IEEE ICCV Workshops*, 2015, pp. 1–23.

[3] A. Singh, A. Roy, X. Zhang, and M. Jagersand, "Modular Decomposition and Analysis of Registration based Trackers," in *CRV*, June 2016.

[4] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," in *7th International Joint Conference on Artificial intelligence*, vol. 2, 1981, pp. 674–679.

[5] G. Bradski, "OpenCV," *Dr. Dobb's Journal of Software Tools*, 2000.

[6] J.-Y. Bouguet, "Pyramidal Implementation of the Lucas Kanade Feature Tracker: Description of the Algorithm," Intel Corporation Microprocessor Research Labs, Tech. Rep., 2000.

[7] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[8] G. D. Hager and K. Toyama, "Xvision: A portable substrate for real-time vision applications," *Computer Vision and Image Understanding*, vol. 69, no. 1, pp. 23–37, 1998.

[9] G. D. Hager and P. N. Belhumeur, "Efficient Region Tracking With Parametric Models of Geometry and Illumination," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 10, pp. 1025–1039, October 1998.

[10] A. Harris and J. Conrad, "Survey of popular robotics simulators, frameworks, and toolkits," in *Southeastcon, 2011 Proceedings of IEEE*, March 2011, pp. 243–249.

[11] A. Singh and M. Jagersand, "Unifying Registration based Tracking: A Case Study with Structural Similarity," 2016, arXiv:1607.04673 [cs.CV].

[12] E. Marchand, F. Spindler, and F. Chaumette, "ViSP for visual servoing: a generic software platform with a wide class of robot control skills," *Robotics Automation Magazine, IEEE*, vol. 12, no. 4, pp. 40–52, Dec 2005.

[13] T. Dick, C. Perez, M. Jagersand, and A. Shademan, "Realtime Registration-Based Tracking via Approximate Nearest Neighbour Search," in *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013.

[14] J. Kwon, H. S. Lee, F. C. Park, and K. M. Lee, "A geometric particle filter for template-based visual tracking," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 36, no. 4, pp. 625–643, 2014.

[15] X. Zhang, A. Singh, and M. Jagersand, "RKLT: 8 DOF real-time robust video tracking combing coarse RANSAC features and accurate fast template registration," in *CRV*. IEEE, 2015, pp. 70–77.

[16] "Eigen Benchmark," http://eigen.tuxfamily.org/index.php?title=Benchmark, accessed: 2016-02-27.

[17] S. Baker and I. Matthews, "Lucas-Kanade 20 Years On: A Unifying Framework," *IJCV*, vol. 56, no. 3, pp. 221–255, Feb 2004.

[18] A. Dame, "A unified direct approach for visual servoing and visual tracking using mutual information," Ph.D. dissertation, University of Rennes, 2010.

[19] R. Szeliski, "Image Alignment and Stitching: A Tutorial," *Foundations and Trends in Computer Graphics and Vision*, vol. 2, no. 1, pp. 1–104, January 2006.

[20] R. Richa, R. Sznitman, R. Taylor, and G. Hager, "Visual tracking using the sum of conditional variance," in *IROS, IEEE/RSJ International Conference on*, Sept 2011, pp. 2953–2958.

[21] G. G. Scandaroli, M. Meilland, and R. Richa, "Improving NCC-based Direct Visual Tracking," in *ECCV*. Springer, 2012, pp. 442–455.

[22] G. H. Rogerio Richa, Raphael Sznitman, "Robust Similarity Measures for Gradient-based Direct Visual Tracking," CIRL, Tech. Rep., June 2012.

[23] G. Silveira and E. Malis, "Real-time visual tracking under arbitrary illumination changes," in *CVPR. IEEE Conference on*, 2007, pp. 1–6.

[24] A. Bartoli, "Groupwise geometric and photometric direct image registration," *PAMI, IEEE Transactions on*, vol. 30, no. 12, pp. 2098–2108, 2008.

[25] H.-Y. Shum and R. Szeliski, "Construction of Panoramic Image Mosaics with Global and Local Alignment," *IJCV*, vol. 36, no. 2, pp. 101–130.

[26] S. Baker and I. Matthews, "Equivalence and efficiency of image alignment algorithms," in *CVPR, IEEE Conference on*, vol. 1, 2001, pp. I–1090–I–1097 vol.1.

[27] S. Benhimane and E. Malis, "Homography-based 2D Visual Tracking and Servoing," *Int. J. Rob. Res.*, vol. 26, no. 7, pp. 661–676, July 2007.

[28] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang, "Incremental Learning for Robust Visual Tracking," *IJCV*, vol. 77, no. 1-3, pp. 125–141, May 2008.

[29] R. P. Woods, S. R. Cherry, and J. C. Mazziotta, "Rapid automated algorithm for aligning and reslicing PET images," *J. Comput. Assist. Tomogr.*, vol. 16, no. 4, pp. 620–633, July 1992.

[30] F. L. Bookstein, "Principal warps: Thin-plate splines and the decomposition of deformations," *PAMI, IEEE Transactions on*, no. 6, pp. 567–585, 1989.

[31] R. Szeliski and J. Coughlan, "Spline-based image registration," *IJCV*, vol. 22, no. 3, pp. 199–218, 1997.

[32] M. Muja and D. G. Lowe, "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration." *VISAPP (1)*, vol. 2, pp. 331–340, 2009.

[33] P. J. Rousseeuw, "Least Median of Squares Regression," *J. Am. Stat. Assoc.*, vol. 79, no. 388, pp. 871–880, 1984.

[34] R. Brooks and T. Arbel, "Generalizing Inverse Compositional and ESM Image Alignment," *IJCV*, vol. 87, no. 3, pp. 191–212, May 2010.