Modular Tracking Framework: A Fast Library for High Precision 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 tracking systems. In order to establish the theoretical basis for its design, a new way to conceptualize registration based trackers is also introduced that decomposes them into three sub modules - Search Method, Appearance Model and State Space Model.**

This framework is not only a practical solution for fast and high precision tracking but 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. Through 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 (**OLT**s) 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. However, these 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 (**RBT**s) (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] that can only perform 2 DOF tracking. Similarly, the ROS library [7] currently does not have any package that implements a modern RBT. 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. This Visual Servoing Platform (ViSP) library [11] does provide somewhat similar tracking functionality as the proposed framework but it has several shortcomings that are enumerated in Sec. V.

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 RBT 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 an RBT 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. This 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 RBT 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], [12]. By following this decomposition closely through 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

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

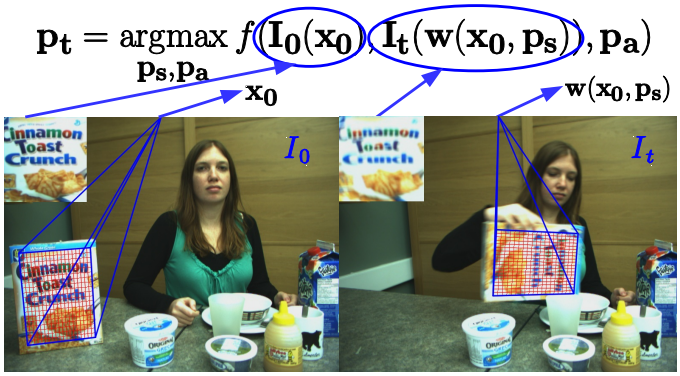

Fig. 1. Two frames from a sequence showing the different components of RBT. The grid of points $\mathbf{x}$ where pixel values are extracted are shown in red and the corresponding patches are in the top left corners. For better visibility, the grid is sampled at 25×25 though higher resolutions may be used in practice.

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 for seamless integration with robotics systems.

Rest of this paper is organized as follows: Sec. II introduces the mathematical basis for the design of MTF while Sec. III describes the class structure of MTF along with specifications for important functions. Sec. IV presents a couple of use cases for MTF followed by performance comparisons with ViSP and OLTs in Sec. V. Sec. VI concludes with ideas for future extensions to MTF.

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 [13] 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 $\mathbf{x_t} = [\mathbf{x_{1t}}, ..., \mathbf{x_{Nt}}] \in \mathbb{R}^{2\times N}$ with $\mathbf{x_{kt}} = [x_{kt}, y_{kt}]^T \in \mathbb{R}^2$ being the coordinates of pixel k.

Further, $\mathbf{w}(\mathbf{x}, \mathbf{p_s}) : \mathbb{R}^{2\times N} \times \mathbb{R}^S \mapsto \mathbb{R}^{2\times N}$ 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 [14].

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) [15], normalized cross correlation (NCC) [16], mutual information (MI) [13] and cross cumulative residual entropy (CCRE) [17]. So far, the only examples with $A \neq 0$, to the best of our knowledge, are those with an illumination model (**ILM**) [18], [19] 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.

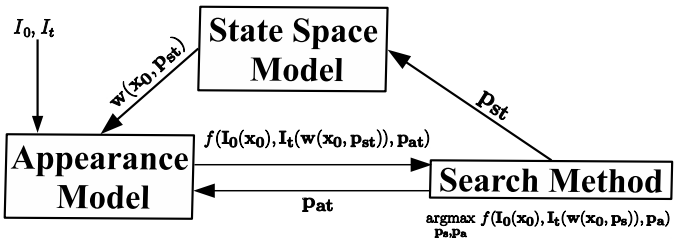

Fig. 2. Decomposition of RBT showing the interaction between the resultant sub modules

B. Registration based tracking

Using this notation, RBT 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,

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

A pictorial representation of the meanings of various components in this equation is shown in Fig. 1. As has been observed before [14], [17], 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.

Fig. 2 shows the effective flow of information between the three sub modules though in practice SM serves as the interface between AM and SSM which do not interact directly. Following steps are performed for each frame I_t:

1) SM computes the optimum parameters for $\mathbf{w}$ and f and passes these respectively to the SSM and AM.
2) SSM then warps the initial grid points $\mathbf{x_0}$ using these parameters and passes the resultant points $\mathbf{w}(\mathbf{x_0}, \mathbf{p_{st}})$ to the AM
3) AM extracts pixel values at these points and computes the similarity of the resultant patch with the template using $\mathbf{p_{at}}$ which it then passes back to the SM.
4) SM uses this similarity to find the parameters $\mathbf{p_{t+1}}$ that maximize it for the next frame I_{t+1}.

C. Gradient Descent and the Chain Rule

Though several types of SMs have been reported in literature, gradient descent (GD) based methods [4] are most widely used due to their speed and simplicity. Included in this category is the LK tracker that can be formulated in four different ways [20] 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 variants are thus called Forward Additive (**FALK**)

[4], Inverse Additive (**IALK**) [9], Forward Compositional (**FCLK**) [21] and Inverse Compositional (**ICLK**) [22]. There is also a more recent approach called Efficient Second order Minimization (**ESM**) [23] that tries to make the best of ICLK and FCLK by using information from both I_0 and I_t.

All of these SMs 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:

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

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.

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

$$\mathbf{H} = \frac{\partial \mathbf{w}}{\partial \mathbf{p}}^T \nabla\mathbf{I}^T \frac{\partial^2 f}{\partial \mathbf{I}^2} \nabla\mathbf{I} \frac{\partial \mathbf{w}}{\partial \mathbf{p}} + \frac{\partial f}{\partial \mathbf{I}} \left(\frac{\partial \mathbf{w}}{\partial \mathbf{p}}^T \nabla^2\mathbf{I} \frac{\partial \mathbf{w}}{\partial \mathbf{p}} + \nabla\mathbf{I} \frac{\partial^2 \mathbf{w}}{\partial \mathbf{p}^2} \right) \tag{4}$$

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. More details can be found in [20] and [24].

D. Stochastic Search

A limitation of GD 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 $\mathbf{p}$. MTF currently has four main SMs in this category - particle filter (**PF**) [25], nearest neighbor (**NN**) [26], RANSAC [27] and Least Median of Squares (**LMS**) [28].

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 [25].

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. 3, 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`. [2] 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 RBTs are well known to be prone to failures and more than three decades of research has failed to make significant improvements in this regard [3], [24], the composite approach seems to be one of the more promising ones [27]. MTF has thus been designed to facilitate work in this direction with several existing trackers of this type described in [24].

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 [24] 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`. This has two further

[2] 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*.

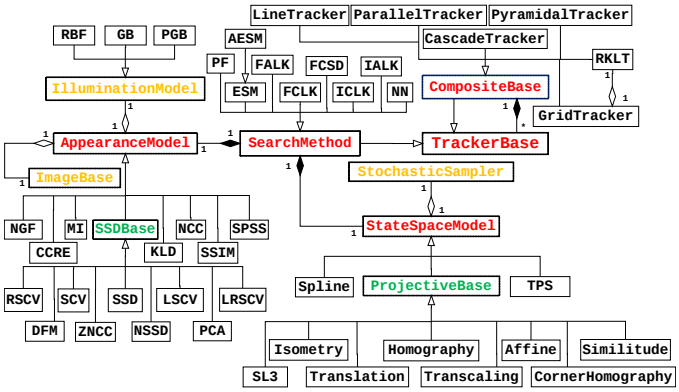

Fig. 3. MTF Class Diagram showing all models currently implemented. Pure and partially abstract classes are respectively shown in red and green while concrete classes are in black. Classes that are sub parts of AM and SSM are in yellow. Meanings of acronyms not defined in text are in [24].

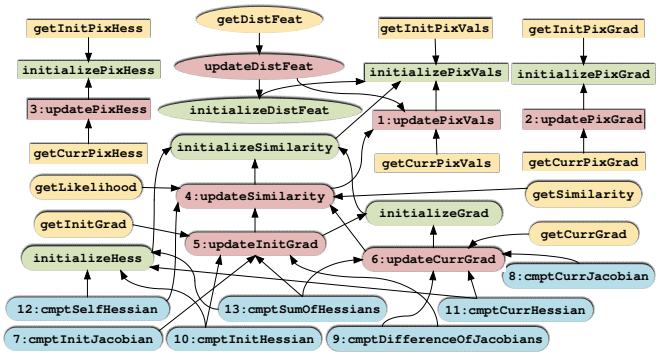

Fig. 4. Dependency relationships between various functions in AM: an arrow pointing from A to B means that A depends on B. Color of a function box denotes its type - green: initializing; red: updating; blue: interfacing and yellow: accessor function. Shape of a function box represents the part of AM it belongs to - rectangle: Image Operations; rounded rectangle: Similarity Functions; ellipse: Distance Feature.

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 sub module. Secondly, it is not necessary for all combinations of AMs and SSMs to support all SMs.

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. 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 $\mathbf{p}$. Detailed specifications for important functions in both classes can be found in [24].

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.

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 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.

2) Similarity Functions: This is the core of AM and handles the computation of the similarity measure $f(\mathbf{I}^*, \mathbf{I}^\mathbf{c}, \mathbf{p_a})$ and its derivatives $\partial f/\partial\mathbf{I}$ and $\partial^2 f/\partial\mathbf{I}^2$ w.r.t. both $\mathbf{I}^*$ and $\mathbf{I}^\mathbf{c}$. 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.

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 [18], [19] 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 $\mathbf{p_a}$ and ILM is only a special case. AM also supports learning to update the object's appearance, as present, for instance, in PCA [29].

Since several of the functions in this part of AM involve common computations, there exist *transitive dependency* relationships between them (Fig. 4) 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 [30] 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.

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. 3) 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.

2) Stochastic Sampler: This is provided to support stochastic SMs. It offers functions to generate small random incremental updates to $\mathbf{p_s}$ (generatePerturbation) by drawing these from a zero mean normal distribution, generate stochastic state samples using random walk (additiveRandomWalk) or first order auto regression (additiveAutoRegression1), estimate the mean of a set of samples of $\mathbf{p_s}$ (estimateMeanOfSamples) and estimate the best fit $\mathbf{p_s}$ from a set of original and warped point pairs (estimateWarpFromPts) using a robust method like RANSAC or LMS.

IV. Use Cases

Algorithm 1 Object Tracking - Simple

```
1: using namespace mtf;
2: ICLK<SSD, Homography> tracker;
3: tracker.initialize(input.getFrame(), init_location);
4: while input.update() do
5:     tracker.update(input.getFrame());
6:     new_location ← tracker.getRegion();
7: end while
```

This section presents following use cases for MTF in C++ style pseudo code:

Algorithm 2 Object Tracking - Composite

```
1: PF<ZNCC, Affine> tracker1;
2: FCLK<SSIM, SL3> tracker2;
3: vector<TrackerBase*> trackers = {&tracker1, &tracker2};
4: CascadeTracker tracker(trackers);
5: lines 3-7 of Alg. 1
```

Algorithm 3 UAV Trajectory Estimation in Satellite Image

```
1: ESM<MI, Similitude> tracker;
2: uav_img_corners←getFrameCorners(input.getFrame());
3: tracker.initialize(satellite_img, init_uav_location);
4: curr_uav_location←tracker.getRegion();
5: while input.update() do
6:     tracker.initialize(input.getFrame(), uav_img_corners);
7:     tracker.setRegion(curr_uav_location);
8:     tracker.update(satellite_img);
9:     curr_uav_location←tracker.getRegion();
10: end while
```

Algorithm 4 Online Image Mosaicing

```
1: FALK<MCNCC, Isometry> tracker;
2: mos_img ← writePixelsToImage(input.getFrame(), init_mos_location, mos_size);
3: mos_location ← init_mos_location;
4: while input.update() do
5:     temp_img ← writePixelsToImage(input.getFrame(), mos_location, mos_size);
6:     tracker.initialize(temp_img, mos_location);
7:     tracker.update(mos_img);
8:     mos_location ← tracker.getRegion();
9:     mos_img ← writePixelsToImage(input.getFrame(), mos_location, mos_size);
10: end while
```

- Track an object in an image sequence using a simple (Alg. 1) and a composite (Alg. 2) tracker.
- Estimate the trajectory of a UAV within a large satellite image of an area from images it took while flying over that area (Alg. 3). It is assumed here that the size of these images is approximately same as that of the corresponding region in the satellite image.
- Create an image mosaic in real time from a video sequence captured by a camera moving over different parts of the planar scene to be stitched (Alg. 4).

Working examples for all use cases are in the supplementary video. Following are some additional details regarding variables and functions used in these algorithms that may assist the reader in understanding them better:

- MTF comes with an input module with wrappers for the image capturing functions in OpenCV, ViSP and XVision. Represented by *input* in these algorithms, it is assumed to have been initialized with the appropriate source.
- Though only five combinations of SMs, AMs and SSMs are shown here, these can be replaced by virtually any combination of methods in Fig. 3.
- MTF provides a set of general utility functions for image and warping related operations, following of which have been used in Alg. 3 and 4:
  - getFrameCorners(*image*) returns a 2×4 matrix containing the corners of *image* and is thus used when the entire image is to be considered as the tracked region.
  - writePixelsToImage(*patch*, *corners*, *size*) writes the pixel values in *patch* to an image with dimensions *size* in the region bounded by *corners*.
- *init_uav_location* in Alg. 3 is the location of the first frame in the UAV sequence within the satellite image *satellite_img*.

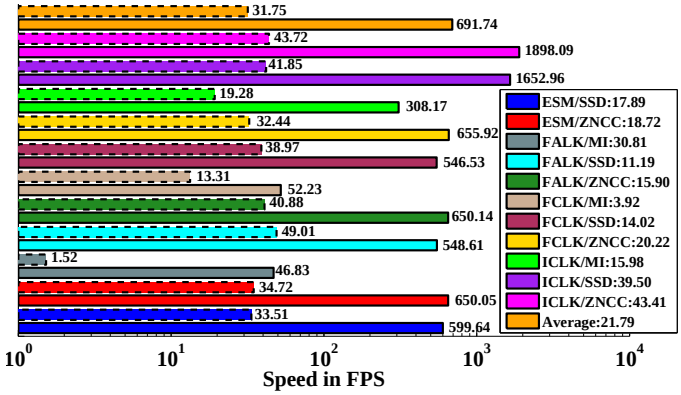

Fig. 5. ViSP vs MTF average tracker speeds for all combinations of SMs and AMs supported by ViSP. MTF and ViSP results are shown in **solid and dotted lines** respectively. Note that **logarithmic scaling** has been used on the x axis for better visibility of ViSP bars though the actual figures are also shown. **Speedup** provided by MTF is shown in the legends. Results were generated on a 4 GHz Intel Core i7-4790K machine with 32 GB RAM.

- *init_mos_location* in Alg. 4 is the location of the first frame in the sequence within the mosaic image of size *mos_size*.

V. Performance

An existing system that is functionally similar to MTF is the template tracker module of the ViSP library [11]. It provides 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, one more GD based SM as well as four stochastic SMs. It also allows multiple SMs to be combined effortlessly to create novel composite SMs. Thirdly, ViSP trackers are significantly buggy and only completed about 70% of the tested sequences.

Lastly, and perhaps most importantly from a practical standpoint, MTF is significantly faster than ViSP. As shown in Fig. 5, MTF is usually more than an order of magnitude faster, with its speed being over 20 times higher than ViSP on average. This is mainly because it uses the Eigen library [31] for all mathematical computations and this is known to be one of the fastest [32]. It may be noted that MTF has not been carefully optimized and parallelized yet and the speed gain is likely to increase further once this is completed. Unlike systems like ViSP and XVision that use their own linear algebra subsystems, being based on an open source library that is under active development affords MTF with the further advantage of automatically improving in speed and reliability whenever improvements are made to this library. This speed advantage is particularly significant in applications like visual servoing and SLAM where several different regions in the same video stream often need to be tracked simultaneously. In addition to allowing them to run in real time, MTF offers another benefit for such multi tracking applications. For each feature or region of the scene, the tracker best matching the characteristics of that region needs to be selected for optimal performance since different combinations of methods are best suited for distinct scenarios [3]. This is much easier to accomplish using MTF than it would be by combining different executables, if any, published by the authors of the respective methods.

This section is concluded with a comparison of MTF trackers with 8 state of the art OLTs [2] to validate the suitability of the former for robotics applications. Results are shown in Fig. 6. As expected, all the OLTs have far lower SR than both 2 and 8 DOF RBTs for smaller t_p since they are less precise in general [2]. They do close the gap as t_p increases but none manage to surpass even the best 2 DOF RBTs, which themselves are notably worse than the 8 DOF ones. This difference is even larger if OLTs and 2 DOF RBTs are evaluated against the full pose ground truth which is more indicative of the motions that need to be tracked in real tasks. Though the superiority of DSST and Struck over other OLTs is consistent with VOT results [2], the very poor performance of GOTURN [33], which is one of the best trackers there, indicates a fundamental difference in the challenges involved in the two paradigms of tracking. The speed plot shows another reason why OLTs are not suitable for high speed tracking applications - they are 10 to 30 times slower than the faster RBTs.

VI. Conclusions and Future Work

This paper presented a modular and extensible open source framework for RBT. It provides highly efficient C++ implementations for several well established trackers that will hopefully address practical tracking needs of the wider robotics community. To this end, it has been designed to integrate well with popular libraries like ROS, OpenCV and ViSP so it can be easily used with existing as well as future projects that require fast trackers. A novel method to decompose RBTs into sub modules was also formulated that can be used to improve the study of this domain of tracking.

MTF is still a work in progress and offers several promising avenues of future extensions for each of the sub modules. This includes novel composite SMs especially those that, like `GridTracker`, run a large number of relatively simple trackers simultaneously and combine their outputs to deduce the state of the tracked patch with greater precision and robustness than any single tracker can possibly provide. A partially implemented work in this direction is the `LineTracker` (Fig. 3) [24] that does not work well yet but can be improved using more robust methods for estimating best fit lines and better constrains between different lines.

One of the most potentially beneficial ways to improve AMs is the incorporation of online learning to update the template. As mentioned in Sec. III-A, MTF is designed to support this and two related AMs - `DFM` [34] and `PCA` [29] - already exist that respectively utilize offline and online learning. They are both rather simple and out dated, however, and more sophisticated learning methods need to be implemented, especially those utilizing deep neural networks

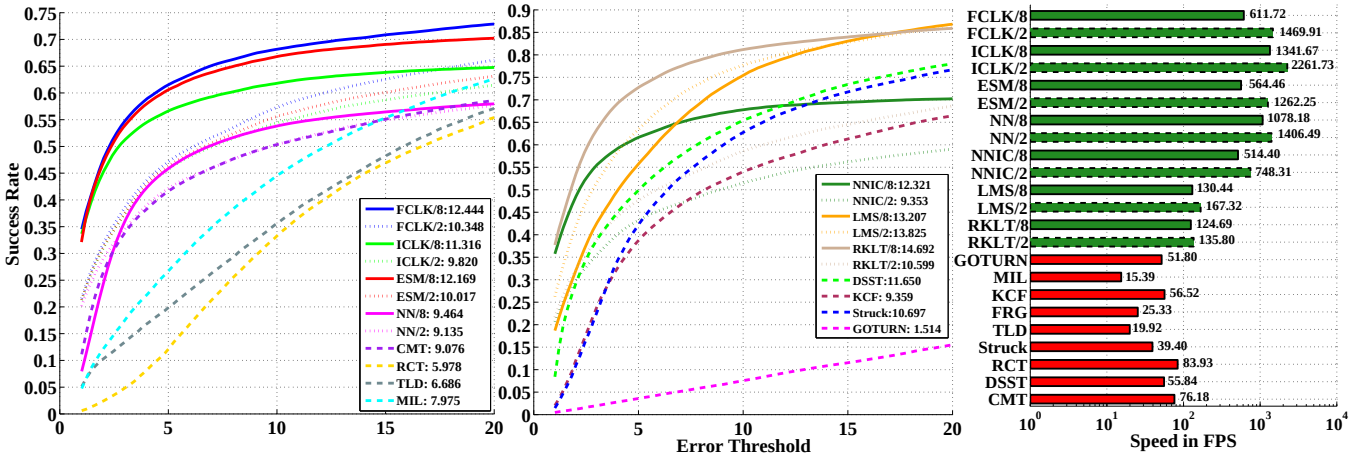

Fig. 6. Comparing OLTs with 2 and 8 DOF RBTs in terms of both accuracy and speed. OLTs are shown in **dashed** lines while 2 and 8 DOF RBTs are in **dotted** and **solid** lines respectively. Accuracy is measured by the tracking success rates (**SR**) for a range of error thresholds (t_p) over 4 large datasets with more than 100000 frames. OLTs and 2 DOF RBTs were evaluated against 2 DOF ground truth for fairness. Legends in the SR plots show the areas under the respective curves. Speed plot on the right has **logarithmic scaling** on the x axis for clarity though actual figures are also shown. Original C++ implementations were used for all OLTs. More details about the evaluation methodology and the meanings of several acronyms can be found in [12].

that have become popular in recent years. ILMs can also be extended with better parameterization that can account for other sources of variations in the appearance of the object patch such as motion blur and occlusion. Another promising extension is the ability to handle depth information from 3D cameras like Kinect whose increasing ubiquity may make this the easiest way to improve tracking performance.

SSMs can be improved by using motion learning from annotated sequences to generate better stochastic samples. It has been seen in Sec. II-D that the performance of stochastic SMs depends largely on the quality of samples so any progress in this direction should definitely be worthwhile. Addition of non rigid SSMs that can go beyond the basic planar projective transforms will also a useful extension albeit with somewhat limited application domain. SSMs that support 3D motion estimation are also needed to complement the aforementioned depth information processing support in AMs. Improvements can also be made to the implementations of existing methods to make them one practically useful. For instance, slower methods like `PF`, `MI`, `CCRE`, `NGF` and `GridTracker` need to be efficiently parallelized.

References

[1] A. W. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah, "Visual tracking: An experimental survey," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 36, no. 7, pp. 1442–1468, 2014.

[2] M. Kristan, A. Leonardis, J. Matas, M. Felsberg, *et al.*, *The Visual Object Tracking VOT2016 Challenge Results*. Cham: Springer International Publishing, 2016, pp. 777–823.

[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] 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.

[12] A. Singh, M. Siam, and M. Jagersand, "Unifying Registration based Tracking: A Case Study with Structural Similarity," 2017, arXiv:1607.04673 [cs.CV], accepted in WACV 2017.

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

[14] R. Szeliski, "Image alignment and stitching: A tutorial," *Foundations and Trends® in Computer Graphics and Vision*, vol. 2, no. 1, pp. 1–104, 2006.

[15] R. Richa, R. Sznitman, R. Taylor, and G. Hager, "Visual tracking using the sum of conditional variance," in *IROS*, Sept 2011, pp. 2953–2958.

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

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

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

[19] A. Bartoli, "Groupwise geometric and photometric direct image registration," *TPAMI*, vol. 30, no. 12, pp. 2098–2108, 2008.

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

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

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

[23] 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.

[24] A. Singh and M. Jagersand, "Modular Tracking Framework: A Unified Approach to Registration based Tracking," 2016, arXiv:1602.09130.
[25] 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 (PAMI), IEEE Transactions on*, vol. 36, no. 4, pp. 625–643, 2014.
[26] 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.
[27] 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*, 2015, pp. 70–77.
[28] P. J. Rousseeuw, "Least Median of Squares Regression," *J. Am. Stat. Assoc.*, vol. 79, no. 388, pp. 871–880, 1984.
[29] 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.
[30] M. Muja and D. G. Lowe, "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration." *VISAPP (1)*, vol. 2, pp. 331–340, 2009.
[31] "Eigen: A C++ template library for linear algebra," http://eigen.tuxfamily.org, accessed: 2017-02-04.
[32] "Eigen Benchmark," http://eigen.tuxfamily.org/index.php?title=Benchmark, accessed: 2017-02-04.
[33] D. Held, S. Thrun, and S. Savarese, *Learning to Track at 100 FPS with Deep Regression Networks*. Cham: Springer International Publishing, 2016, pp. 749–765.
[34] M. Siam, "CNN Based Appearance Model with Approximate Nearest Neigbour Search," University of Alberta, Tech. Rep., 2015. [Online]. Available: http://webdocs.cs.ualberta.ca/~vis/mtf/dfm_report.pdf