Appearance learning for 3D tracking of robotic surgical tools

Austin Reiter, Peter K Allen and Tao Zhao

The International Journal of Robotics Research published online 11 November 2013
DOI: 10.1177/0278364913507796

Article

The International Journal of
Robotics Research
0(0) 1–15
© The Author(s) 2013
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/0278364913507796
ijr.sagepub.com

Appearance learning for 3D tracking of robotic surgical tools

Austin Reiter[1], Peter K Allen[1] and Tao Zhao[2]

Abstract

In this paper, we present an appearance learning approach which is used to detect and track surgical robotic tools in laparoscopic sequences. By training a robust visual feature descriptor on low-level landmark features, we build a framework for fusing robot kinematics and 3D visual observations to track surgical tools over long periods of time across various types of environment. We demonstrate 3D tracking on multiple types of tool (with different overall appearances) as well as multiple tools simultaneously. We present experimental results using the da Vinci® surgical robot using a combination of both ex-vivo and in-vivo environments.

Keywords

Tool tracking, surgical robotics, learning features, fusion

1. Introduction

Advancements in minimally invasive surgery have come about through technological breakthroughs in endoscopic technology, smarter instruments, and enhanced video capabilities (Mack, 2001). These achievements have had a common goal of continuing to reduce the invasiveness of surgical procedures. Robotic hardware and intelligent algorithms open the doors to more complex procedures by enhancing the dexterity of the surgeon's movements as well as increasing safety through mechanisms like motion scaling and stereo imaging.

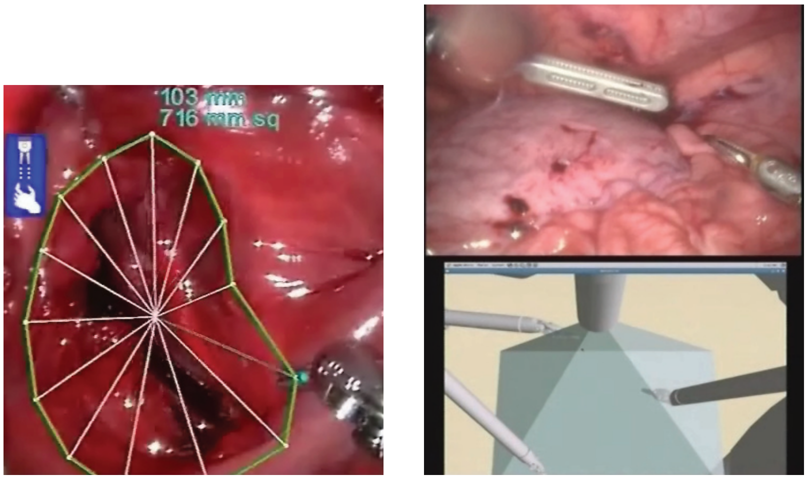
Intuitive Surgical's da Vinci® robot (Intuitive Surgical, 1995) is the most prevalent example of such a technology, as there are more than 1800 da Vinci® surgical systems in operating rooms worldwide which performed about 360,000 procedures in 2011. In this system, high-definition stereo vision delivers a perceptual 3D image to the surgeon which helps to see the anatomy and interact with the surgical tools with great clarity. Augmenting the surgeon's vision with other relevant information in the form of graphical overlays can further help the surgeons/patients in a different dimension. Tool tracking is a manifestation of intelligent computation which can improve the situational awareness for a surgeon during a procedure.

Knowledge of the locations of tools in the endoscopic image can enable a wide spectrum of applications. Accurate tool localizations can be used as a *virtual ruler* (Leven et al., 2005) (see Figure 1(a)), capable of measuring the distances between various points in the scene, such as the sizes of anatomical structures. Graphical overlays can indicate the status of a particular tool, for example in the case of the firing status of an electro-cautery tool. These indicators can be placed at the tip of the tool in the visualizer which is close to the surgeon's visual center of attention, enhancing the overall safety of using such tools. It can also be useful in managing the tools that are off the screen (Malpani et al., 2011) (see Figure 1(b)), increasing the patient's safety, or for visual servoing of motorized cameras.

The joints of a robotic surgical system are typically equipped with encoders so that the pose of the end effectors can be computed using forward kinematics. In the da Vinci®, the kinematic chain between the camera and the tool tip involves 18 joints and more than two meters in cumulative length, which is challenging to the accuracy of absolute position sensing and would require arduous and time-consuming procedures to accurately calibrate. However, a master–slave robotic system does not require high absolute accuracy because humans are in the control loop. As a result, we have observed up to one inch of absolute error, which is too large for most of the applications that are mentioned above. Therefore, tracking the tools from images is a practical and non-invasive way to achieve the accuracy requirements of the applications.

[1]Department of Computer Science, Columbia University, USA
[2]Intuitive Surgical, Inc., CA, USA

Corresponding author:
Austin Reiter, 500 W. 120th Street, M.C. 0401 New York, NY 10027, USA.
Email: areiter@cs.columbia.edu



(a) Ruler (b) Lost tool

Fig. 1. Two applications of tool tracking: in (a), a picture of the measurement tool measuring the circumference and area of a mitral valve is shown. In (b), an example scenario of a lost tool (e.g. outside the camera's field of view) is shown, whereby the endoscopic image (top) shows only two tools, and with corrected kinematics and a graphical display (bottom), we can accurately show the surgeon where the third tool (out of the bottom left corner) is located and posed so they can safely manipulate the tool back into the field of view.

In this paper we present a tracking system which learns classes of natural landmarks on articulated tools off-line by training an efficient multi-class classifier on a discriminative feature descriptor from manually ground-truthed data. We run the classifier on a new image frame to detect all extrema representing the location of each feature type, where confidence values and geometric constraints help to reject false positives. Next, we stereo match in the corresponding camera to recover 3D point locations on the tool. By knowing a priori the locations of these landmarks on the tool part (from the tool's computer-aided design model), we can recover the pose of the tool by applying a fusion algorithm of kinematics and these 3D locations over time and computing the most stable solution of the configuration. Our tracker is able to deal with multiple tools simultaneously by applying a tool association algorithm and is able to detect features on different types of tool. This work is an extension of that presented in Reiter et al. (2012a), where only one tool type is dealt with in a single-tool tracking approach. The contributions of the current paper are to extend the learning system to multiple tool types and multiple tools tracked simultaneously, as well as demonstrating the system across various types of surgical data. More details on each of these steps follow in the remaining sections.

1.1. Prior work

There has been much progress in the field of tracking surgical instruments. Typically either color or texture is used, and in cases where information about the tool is known a priori, a shape model can be used to confine the search space (Doignon et al., 2006; Voros et al., 2007; Pezzementi et al., 2009). A common method is to design a custom marker, as in Wei et al. (1997a,b) and Groeger et al. (2008), to assist in tool tracking. Here, the authors argue that geometry is not reliable enough for tracking, and a color marker is designed by analyzing the hue/saturation/value (HSV) color space to determine what color components are not common in typical surgical imagery. Next, the authors fabricate their own custom marker to be placed on the tool. A training step creates a kernel classifier which can then label pixels in the frame as either foreground (tool) or background. Similarly, the authors in Zhang and Payandeh (2002) design a marker with three stripes that traverse the known diameter of the tool which allows the estimation of depth information of the tool's shaft from the camera. An alternative example of a marker designed as a bar code is described in Zhao et al. (2009a,b).

Color may be exploited without custom markers, as in Lee et al. (1994), in which the authors use different color signatures of organs and instruments to classify individual pixels by training on a large sample of pixels from endoscopic sequences. A Bayesian classifier maximizes the a posteriori probability of the class assignment in order to distinguish organ pixels from instrument pixels. Often, simple assumptions can be made about the environment, such as determining 'gray' regions and labeling them as the instrumentation (Doignon et al., 2004, 2005, 2006). The authors contribute a new definition of color purity component and attempt to extract boundaries of nearly uniformly gray regions to develop the idea that the color saturation is the most discriminating attribute for gray region segmentation, and in so doing, define a new definition of saturation.

Another technique to aid in tracking is to affix assistive devices to the imaging instrument itself. In Krupa et al. (2003), a laser-pointing instrument holder is used to project laser spots into the laparoscopic imaging frames. This is useful for when the tools move out of the field of view of the camera. The laser pattern projected onto the organ surface provides information about the relative orientation of the instrument with respect to the organ. Optical markers are used on the tip of the surgical instruments, and these markers used in conjunction with the image of the projected laser pattern allow for measurements of the pointed organ and the instrument.

Prior information of the surgical tools may be used to confine the search space for the instrument (Voros et al., 2007) and detect the shaft from the insertion point. Here, the authors perform a calibration step to define the 3D insertion point of the instrument into the abdominal cavity. This gives shape considerations to confine the search space for the instrument and helps achieve real-time processing in order to fit a cylinder to the tool's shaft. In Wolf et al. (2011), the abdominal wall is parameterized as a spherical grid using the known insertion point. A discretized, hexagonal geode is constructed where each hexagon represents a candidate pose of the tool through the insertion point and a particle filter determines the most likely pose as a pan/tilt from the insertion point.

Off-line learning has been used to combine multiple features together into a strong feature framework (Pezzementi et al., 2009), wherein the authors extract color and texture features and train off-line on manually labeled training images. Every pixel is labeled as one of three classes (shaft, metal, and background) and class-conditional probabilities are assigned to each pixel. The object configuration is estimated by using a prior geometric model of the object and maximizing the correlation between the rendering and the probability map. A similar, more recent, approach (Allan et al., 2013) used a random-forest classifier with a combination of different features to label pixels which belong to surgical tools in order to estimate the pose. Although the approach is not real-time, the method reinforces the merits of using a classifier over a multi-feature framework to robustly detect surgical tools.

Previously, we have used on-line learning (Reiter and Allen, 2010) to combine multiple features into a composite likelihood framework. In this work, probability maps from several independent features along with lower-level corner features are used to learn new parts of the tool as it moves in the scene. The low-level features grow into the likelihood maps to discover new parts of the tool. This requires minimal up-front information and can track for long periods of time by adjusting to the appearance of the tool over time.

Template matching is also a popular technique, as in Burschka et al. (2004), where a 2D image template is used to keep track of the da Vinci® tool tip and stereo matching in the corresponding camera to localize a single 3D point representing the tool's centroid. Our recent template matching work was presented in Reiter et al. (2012c), which created templates on-line using a robotic graphical renderer. The templates were created using different kinematic configurations of the robot near the current raw kinematics estimate, and the solution was refined by matching gradient orientation templates to the real image for a more accurate kinematic configuration.

2. Methods

In this section we present an overview of our tool tracking method. Figure 2 shows a visual overview of our detection and tracking system. Before we begin, we present an overview of the robotic hardware system as well as information on calibration procedures performed prior to the work presented in this paper.

2.1. System overview

The da Vinci® surgical robot is a tele-operated, master–slave robotic system. The main surgical console is separated from the patient: the surgeon sits in a stereo viewing console and controls the robotic tools with two master tool manipulators (MTMs) while viewing stereoscopic high-definition video. The patient-side hardware contains three robotic manipulator arms along with an endoscopic robotic arm for the stereo laparoscope. A typical robotic arm has seven total degrees of freedom (DOFs), and articulates at the wrist. The stereo camera system is calibrated for both intrinsics and stereo extrinsics using standard camera calibration techniques (Zhang, 2000), and all images are rectified for lens distortion when processed with the methods in this paper. The cameras on the robot have the ability to adjust focus, yielding non-constant camera calibration configurations. To deal with this, we perform camera calibration at several different discrete focus settings off-line (once), and then linearly interpolate the calibration parameters based on any given focus setting on-line to provide stereo calibration parameters at all times during a procedure.

2.2. Scene labeling

We begin with the method described in Pezzementi et al. (2009) to label every pixel in the image as one of three classes: metal, shaft, or background (listed as module A in the algorithm overview of Figure 2). A Gaussian mixture model (GMM) (Duda et al., 2001) of several color and texture features is learned off-line for each of these three classes. Subsequently, we can assign a class-conditional probability for each of the classes to every pixel and assign a label. Figure 3 shows an example result of this pixel labeling routine, with the original image from an in-vivo porcine sequence on the upper left, the metal class on the upper right, the shaft class on the lower left, and the background class on the lower right. The metal class represents all pixels located at the distal tip of the tool, from the clevis to the grippers. These are where all of the features which we wish to detect are located. Additionally, we will describe later on how the shaft class is used to fit a cylinder to the tool's shaft, whenever possible.

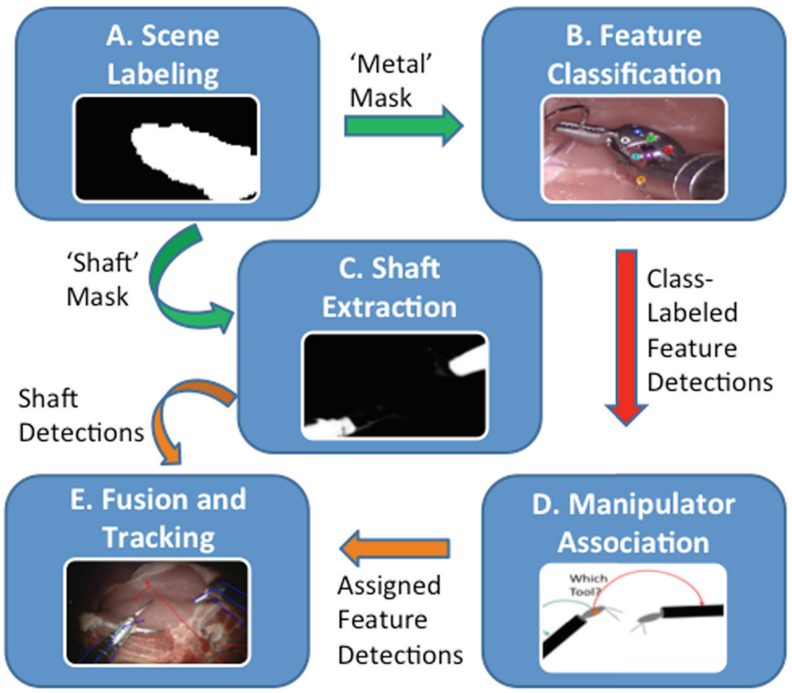

A. Scene Labeling
'Metal' Mask
B. Feature Classification
'Shaft' Mask
C. Shaft Extraction
Class-Labeled Feature Detections
Shaft Detections
E. Fusion and Tracking
Assigned Feature Detections
D. Manipulator Association
Which Tool?

Fig. 2. Algorithm overview of the tracking system. A: The *scene labeling* module applies a multi-feature training algorithm to label all pixels in the image as one of three classes: metal, shaft, and background, producing binary masks for each. B: The *feature classification* module uses a classifier on feature descriptors to localize known landmarks on the tool tips. C: The *shaft extraction* uses the shaft mask from module A to fit cylinders to the shaft pixels in the image for all visible tools, whenever possible. D: The *patient-side manipulator association* module uses class-labeled feature detections output from module B to determine which feature is associated with which tool in the image. E: The *fusion and tracking* module takes outputs from both C and D to fuse visual observations with raw kinematics and track the articulated tools over time.

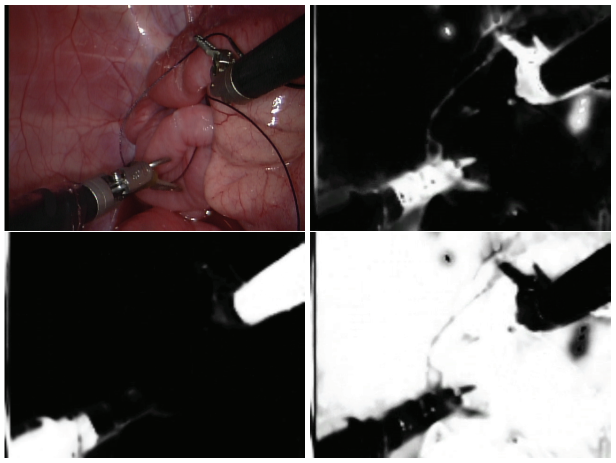


Fig. 3. Example likelihood images from class-conditional pixel labeling as described in Section 2.2. Upper left: the original image from an in-vivo sequence of two robotic tools performing a suturing procedure. Upper right: metal likelihood (e.g. tool tip, clevis). Lower left: tool shaft likelihood. Lower right: background class likelihood.

Typically, surgeries performed with the da Vinci$^{®}$ are quite zoomed in, and so the shaft is not usually visible enough to fit a cylinder (the typical approach to many tool tracking algorithms; Voros et al., 2007). However, at times the camera is zoomed out and so this scene pixel labeling routine allows the algorithm to estimate the 6-DOF pose of the shaft as additional information (Section 2.5). By estimating the approximate distance of the tool from the camera using stereo matching of sparse corner features on the tool's tip, we can estimate if the shaft is visible enough to attempt to fit a cylinder. When the camera is zoomed out, although the shaft is visible the features on the tool tip are not so easily detected. Therefore, we can pick and choose between shaft features, tool-tip features, and a hybrid in between depending on the distance of the tool from the camera. These pixel labelings help to assist in both feature detection and shaft detection, as described further in the following text.

2.3. *Feature classification*

Our feature classification (module B in Figure 2) works by analyzing only the pixels which were labeled as metal, using the method previously described in Section 2.2. This reduces both the false positive rate as well as the computation time, helping us to avoid analyzing pixels which are not likely to be one of our features of interest (because we know beforehand they are all located on the tool tip). We train a

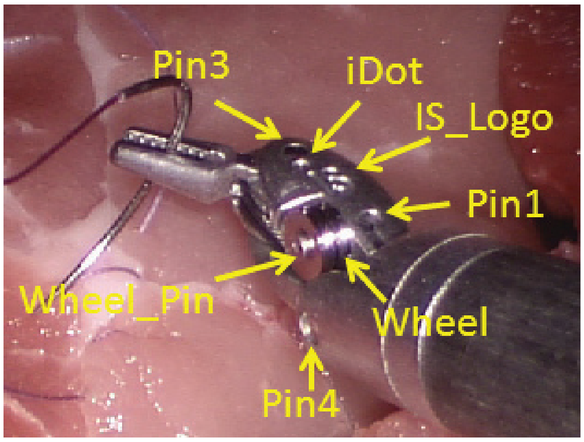


Fig. 4. Ground truth guide for the feature classes we detect on the large needle driver tool. We concentrate on seven different naturally occurring landmarks.

multi-class classifier using a discriminative feature descriptor and then localize class-labeled features in the image. Next, we stereo match and triangulate these candidate feature detections to localize as 3D coordinates. These feature detection candidates are analyzed further using known geometric constraints (described in Section 2.4.2) to remove outliers and then are fed into the fusion and tracking stage of the algorithm. We begin with a detailed description of each of these feature classification steps.

2.3.1. Training data collection We begin by collecting data for the purposes of training our classifier. We use nine different video sequences which span various in-vivo experiments to best cover a range of appearance and lighting scenarios. For training, we use only the large needle driver (LND) tool, however, as we will show later on this will extend well to other types of tool, such as the Maryland bipolar forceps (MBF) and round tip scissors (RTS). Seven naturally occurring landmarks are manually selected and shown in Figure 4 overlaid on an image of the LND. The features chosen are of the pins that hold the distal clevis together, the **IS** logo in the center, and the wheel and wheel pin. For purposes of this paper, from time to time we may refer to this combination of landmarks as a marker pattern, M_i. We also add known invariant locations on the mid-line of the shaft axis (described in Section 2.5) to this marker pattern to be used in the fusion module.

For each frame in the ground truth procedure, we manually drag the best encompassing bounding box around each feature of interest, as we want to avoid contamination from pixels which do not belong to the tool. To obtain as large a dataset as possible with reasonable effort, we coast through small temporal spaces using Lucas–Kanade (KLT) optical flow (Lucas and Kanade, 1981) to predict ground truth locations between user clicks as follows:

1. The user drags a bounding box around a feature of interest.
2. The software uses KLT optical flow to *track* this feature from frame to frame (keeping the same dimensions for the box).
3. As the user inspects each frame, if either the track gets lost or the size changes, the user drags a new bounding box and starts again until the video sequence ends.

This allows for faster ground truth data collection while still manually inspecting for accurate data. Overall, we use $\sim 20,000$ total training samples across the seven feature classes. Before we describe the classifier algorithm, we first discuss the feature descriptor which is used to best discriminate these feature landmarks from each other robustly.

2.3.2. Feature descriptor We require a discriminative and robust region descriptor to describe the feature classes because each feature is fairly small (17–25 pixels wide, or $\sim$ 2% of the image). We choose the region covariance descriptor (Tuzel et al., 2006), where the symmetric square covariance matrix of d features in a small image region serves as the feature descriptor (see Figure 5). Given an image I of size $[W \times H]$, we extract $d = 11$ features, resulting in a $[W \times H \times d]$ feature image:

$$\mathbf{F} = \left[x\ y\ Hue\ Sat\ Val\ I_x\ I_y\ I_{xx}\ I_{yy}\ \sqrt{I_x^2 + I_y^2}\ \arctan\left(\frac{I_y}{I_x}\right) \right] \quad (1)$$

where x, y are the pixel locations; *Hue, Sat, Val* are the hue, saturation, and luminance values from the HSV color transformation at pixel location (x, y); I_x, I_y are the first-order spatial derivatives; I_{xx}, I_{yy} are the second-order spatial derivatives; and the latter two features are the gradient magnitude and orientation, respectively. The first two pixel location features are useful because their correlation with the other features are present in the off-diagonal entries in the covariance matrix (Tuzel et al., 2006). The $[d \times d]$ covariance matrix $C_\mathbf{R}$ of any arbitrary rectangular region $\mathbf{R}$ within the feature image $\mathbf{F}$ (described in equation (1)) then becomes our feature descriptor.

Each $C_\mathbf{R}$ can be computed efficiently using integral images (Viola and Jones, 2004). We compute the sum of each feature dimension as well as the sum of the multiplication of every two feature dimensions. Given these first- and second-order integral image tensors, it can be shown that the covariance matrix of any rectangular region can be extracted in $O(d^2)$ time (Tuzel et al., 2006). Using the ground truth data from Section 2.3.1, we extract covariance descriptors of each training feature and store the associated feature label for training a classifier. However, the d-dimensional nonsingular covariance matrix descriptors cannot be used as is to perform classification tasks directly because they do not lie on a vector space, but rather on a connected Riemannian manifold, and so the descriptors must be post-processed.

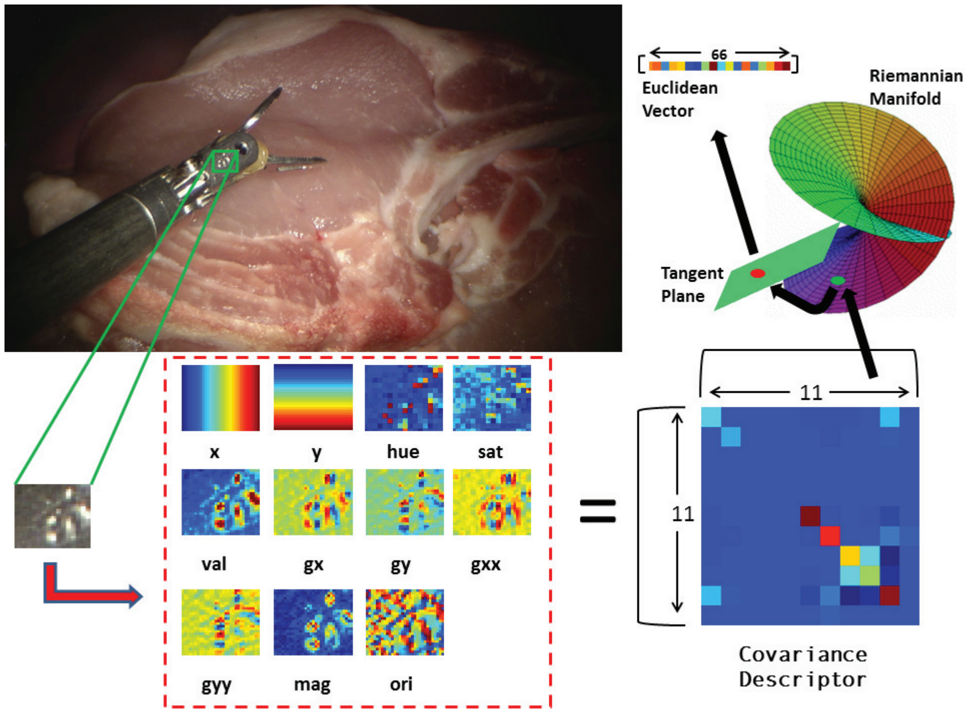


Fig. 5. Several independent features are combined compactly into a single feature descriptor: we use 11 features overall (shown in the red dashed box), specifically the (x,y) locations, HSV color measurements, first- and second-order image gradients, and gradient magnitude and orientation. A rectangular region (green box shown zoomed from the original image at the top) of the image is described by using the covariance matrix of these 11 features within that region, yielding an 11 × 11 symmetric matrix. In order to use this matrix as a descriptor with typical linear mathematical operations, we must map this matrix from its natural Riemannian space to a vector space using Lie algebra techniques (top right), yielding a 66-dimensional vector-space descriptor, described in more details in Sections 2.3.2 and 2.3.3.

2.3.3. Post-processing the covariance descriptors An in-depth mathematical derivation for how to post-process the covariance descriptors to a vector space is shown in Tuzel et al. (2007). Here we briefly summarize the procedure using the same notation. Symmetric positive definite matrices, to which our nonsingular covariance matrices belong, can be formulated as a connected Riemannian manifold (Pennec et al., 2006). A manifold is locally similar to a Euclidean space, and so every point on the manifold has a neighborhood in which a homeomorphism can be defined to map to a tangent vector space.

Our goal is to map our $[d \times d]$ dimensional matrices to a tangent space at some point on the manifold, which will transform the descriptors to a Euclidean multi-dimensional vector space for use within our classifier. Given a matrix $\mathbf{X}$, we define the manifold-specific exponential mapping at the point $\mathbf{Y}$ as

$$\exp_{\mathbf{X}}(\mathbf{Y}) = \mathbf{X}^{\frac{1}{2}} \exp\left(\mathbf{X}^{-\frac{1}{2}}\mathbf{Y}\mathbf{X}^{-\frac{1}{2}}\right)\mathbf{X}^{\frac{1}{2}} \tag{2}$$

and similarly for the logarithmic mapping:

$$\log_{\mathbf{X}}(\mathbf{Y}) = \mathbf{X}^{\frac{1}{2}} \log\left(\mathbf{X}^{-\frac{1}{2}}\mathbf{Y}\mathbf{X}^{-\frac{1}{2}}\right)\mathbf{X}^{\frac{1}{2}} \tag{3}$$

In these formulations, exp and log are the ordinary matrix exponential and logarithmic operations. Finally, we define an orthogonal coordinate system at a tangent space with the vector operation. To obtain the vector-space coordinates at $\mathbf{X}$ for manifold point $\mathbf{Y}$, we perform the following operation:

$$\mathbf{vec}_{\mathbf{X}}(\mathbf{Y}) = upper\left(\mathbf{X}^{-\frac{1}{2}}\mathbf{Y}\mathbf{X}^{-\frac{1}{2}}\right) \tag{4}$$

where *upper* extracts the vector form of the upper triangular part of the matrix. In the end, we are left with a vector space with dimensionality $q = d(d+1)/2$.

The manifold point at which we construct a Euclidean tangent space is the mean covariance matrix of the training data. If we consider $\{\mathbf{X}_i\}_{i=1\ldots N}$ to be the set of points on a Riemannian manifold $\mathfrak{M}$, then to compute the mean matrix $\mu_{\mathbf{C_R}}$ in the Riemannian space, we minimize the sum of squared distances:

$$\mu_{\mathbf{C_R}} = \underset{\mathbf{Y}\in\mathfrak{M}}{\operatorname{argmin}} \sum_{i=1}^{N} d^2(\mathbf{X}_i, \mathbf{Y}) \tag{5}$$

This can be computed using the following update rule in a gradient descent procedure:

$$\mu_{\mathbf{C_R}}^{t+1} = \exp_{\mu_{\mathbf{C_R}}^t}\left[\frac{1}{N}\sum_{i=1}^{N}\log_{\mu^t}(\mathbf{X_i})\right] \tag{6}$$

We use the logarithmic mapping of **Y** at $\mu_{\mathbf{C_R}}$ to obtain our final vectors as in Tuzel et al. (2007). The training covariance matrix descriptors are mapped to this Euclidean space and are used to train the multi-class classifier, described next.

2.3.4. Randomized tree classification There are many multi-class classifiers which may suit this problem, however, runtime is an important factor in our choice of learning algorithm. To this end, we adapt a method called *randomized trees* (RTs) (Lepetit and Fua, 2006) to perform our multi-class classification. In addition to providing feature labels, we would like to retrieve confidence values for the classification task which will be used to construct class-conditional likelihood images for each class. We previously performed a study of different feature descriptors (e.g. scale-invariant feature transforms (SIFT) (Lowe, 2004), histograms of oriented gradients (HoGs) (Dalal and Triggs, 2005), and the covariance descriptors previously described) paired with various classification algorithms (e.g. support vector machines (SVMs) (Cortes and Vapnik, 1995) and two variants on RTs, described next) in Reiter et al. (2012b). In this work, we determined that using the covariance descriptor as the feature descriptor for our chosen landmarks paired with our adaptation of the RT classifier achieves a sufficient level of accuracy and speed for our tool tracking task.

RTs naturally handle multi-class problems very efficiently while retaining an easy training procedure. The RT classifier Λ is made up of a series of L randomly generated trees $\Lambda = [\gamma_1, \ldots, \gamma_L]$, each of depth m. Each tree γ_i, for $i \in 1, \ldots, L$, is a fully balanced binary tree made up of internal nodes, each of which contains a simple, randomly generated test that splits the space of data to be classified, and leaf nodes which contain estimates of the posterior distributions of the feature classes.

To train the tree, the training features are dropped down the tree, performing binary tests at each internal node until a leaf node is reached. Each leaf node contains a histogram of length equal to the number of feature classes B, which in our problem is seven (for each of the manually chosen landmarks shown in Figure 4). The histogram at each leaf counts the number of times a feature with each class label reaches that node. At the end of the training session, the histogram counts are turned into probabilities by normalizing the counts at a particular node by the total number of hits at that node. A feature is then classified by dropping it down the trained tree, again until a leaf node is reached. At this point, the feature is assigned the probabilities of belonging to a feature class depending on the posterior distribution stored at the leaf from training.

Because it is computationally infeasible to perform all possible tests of the feature, L and m should be chosen so as to cover the search space sufficiently and to best avoid random behavior. In this work, we used $L = 60$ trees each of depth $m = 11$. Although this approach has been shown to be very successful for matching image patches (Lepetit and Fua, 2006), traditionally the internal node tests are performed on a small patch of the luminance image by randomly selecting two pixel locations and performing a binary operation (less than, greater than) to determine which path to take to a child. In our problem, we are using feature descriptor vectors rather than image patches, and so we must adapt the node tests to suit our problem.

To this end, we use a similar approach to Bosch et al. (2007) in creating node tests for feature descriptor vectors. In our case, for each internal tree node we construct a random linear classifier h_i to feature vector **x** to split the data,

$$h_i = \begin{cases} \mathbf{n}^{\mathrm{T}}\mathbf{x} + z \le 0 & \text{go to right child} \\ \textit{otherwise} & \text{go to left child} \end{cases} \tag{7}$$

where **n** is a randomly generated vector of the same length as feature **x** with random values in the range $[-1, 1]$, and $z \in [-1, 1]$ is also randomly generated. This test allows for robust splitting of the data and is efficiently utilized as it is only a dot product, an addition, and a binary comparison per tree node. In this way, we train the tree with vectorized versions of the covariance descriptors and build up probability distributions at the leaf nodes. The resulting RT classifier Λ is our final multi-class classifier. The results from each tree γ_i are averaged across all L trees. However, we choose relatively small values for L and m for computation purposes, but the search space is still quite large given the appreciable number of choices for randomly created linear dot products at the internal tree nodes, and this leaves the training approach susceptible to randomness. To alleviate this, we modify the approach further.

2.3.5. Best weighted RTs We developed a method (Reiter et al., 2012b) which is able to improve on the standard RT approach, which we call *best weighted RTs*. The modification lies in two observations:

1. Each tree γ_i is essentially a weak classifier, but some may work better than others, and we can weight them according to how well they behave on the training data.
2. Because of the inherent randomness of the algorithm and the large search space to be considered, we can show improvement by initially creating an *RT bag* Ω of size $E \gg L$. This allows us to initially consider a larger space of trees, but we then evaluate each tree in Ω on the training data in order to select the best L trees for inclusion in the final classifier according to an error metric.

The latter point allows us to consider more of the parameter space when constructing the trees while retaining the computational efficiency of RTs by only selecting the best performers. In order to evaluate a particular tree on the training data, we look at the posterior probability distributions at the leaf nodes. First, we split the training data into *training* and *validation* sets (we typically use $\sim 70\%$ to train and the rest to validate). Next, all trees from the training set in Ω are trained as usual. Given a candidate trained tree $\tilde{\gamma}_i \in \Omega$, we drop each training sample from the validation set through $\tilde{\gamma}_i$ until a leaf node is reached. Given training feature $\mathbf{X_j}$ and feature classes $1, \ldots, B$, the posterior distribution at the leaf node contains b conditional probabilities $p_{\tilde{\gamma}_i}(y|\mathbf{X}_j)$ where $y \in 1, \ldots, B$. To evaluate the goodness of tree $\tilde{\gamma}_i$ on $\mathbf{X_j}$, we compare $p_{\tilde{\gamma}_i}(y_j|X_j)$ to the desired probability $\mathbf{1}$ of label y_j, and accumulate the root-mean squared (RMS) error of all training features $\mathbf{X_j}$ across all validation trees in Ω. The top L trees (according to the lowest RMS errors) are selected for the final classifier Λ. Our initial bag size for this work was $E = 125,000$ candidate tree classifiers, cut down to $L = 60$ trained trees for the final classifier.

In addition to selecting the best trees in the bag, we use the error terms as weights on the trees. Rather than allowing each tree to contribute equally to the final averaged result, we weight each tree as *one over RMS* so that trees that label the validation training data better have a larger say in the final result than those which label the validation data worse. As such, for each $\gamma_i \in \Lambda$ we compute an associated weight w_i such that

$$w_i = \frac{1}{rms_i} \tag{8}$$

where rms_i is the accumulated RMS error of tree γ_i on the validation data. At the end, all weights w_i for $i \in 1, \ldots, L$ are normalized to sum to one and the final classifier result is a weighted average using these weights.

2.3.6. *Feature class labeling*

Given our trained classifier Λ, we detect features for each class label on a test image by computing dense covariance descriptors $\mathbf{C_R}$ (at many locations in the image) using the integral image approach for efficient extraction. Each $\mathbf{C_R}$ is mapped to a vector space using the mean covariance $\mu_{\mathbf{C_R}}$ of the training data as previously described, producing a Euclidean feature $\mathbf{c}_j$. We drop each $\mathbf{c}_j$ through the trees γ_i and average the probabilities at the obtained leaf nodes to get a final probability distribution p_b, representing the probability of $\mathbf{c}_j$ belonging to each of the B feature classes. This results in B class-probability images. To get the pixel locations, we perform non-maximal suppression in each class-probability image.

The reason we use the probabilities instead of the classification labels is that a classification of label b arises because its confidence is greater than all other $B - 1$ classes in the classifier, however, a confidence of 95% for one pixel location means more than a confidence of 51% for that same labeling at a different location. In this case, we would choose the pixel with the higher probability (even given they both have the same label), and for this reason we detect in probability space rather than in labeling space.

2.3.7. *Stereo matching*

Now that we have candidate pixel locations for each feature class, we stereo match the feature detections in the corresponding stereo camera using normalized cross-correlation checks along the epipolar line and triangulate the features to retrieve 3D locations. Using integral images of summations and squared-summations we can efficiently compute correlation windows along these epipoles. However, at this point we only have 3D point locations (in the camera's coordinate system) and associated feature labels, but we do not know with which tool each feature is associated. Next we describe the tool association procedure.

2.4. *Patient-side manipulator association*

At this point, we have class-labeled 3D feature locations, but with multiple tools in the scene it is unclear which feature is associated with which tool. Typically, the da Vinci® has three patient-side manipulators (PSMs), only two of which are visible in the camera frame at any time. We label each manipulator as PSM_0, PSM_1, and PSM_2. For the purposes of this work we only consider two tools simultaneously, PSM_0 and PSM_1, and our goal is to associate feature detections with PSMs (module D in Figure 2).

2.4.1. *Pre-processing the marker patterns*

Each PSM has a marker pattern, M_0 and M_1 respectively, each in their zero-coordinate frame (i.e. the coordinate system before any kinematics are applied to the marker). Using the forward kinematics estimate from each PSM, we rotate the marker patterns to achieve the estimated orientations of each PSM. Note that we do not apply the full rigid-body transform from the forward kinematics because most of the error is in the position, and although the rotation is not fully correct, it is typically close enough to provide the geometric constraints we require. This leaves us with

$$\tilde{M_0} = Rot_0(M_0) \tag{9}$$

$$\tilde{M_1} = Rot_1(M_1) \tag{10}$$

where Rot_0 and Rot_1 are the 3×3 rotation matrices from the full rigid-body transformations representing the forward kinematics for PSM_0 and PSM_1, respectively. Given $\tilde{M_0}$ and $\tilde{M_1}$, we compute 3D unit vectors between each of the rotated point locations within each marker. This yields 7×7 3D unit vectors in a $7 \times 7 \times 3$ matrix for each rotated marker pattern. Additionally, we compute a 7×7 distance matrix D_m between each marker location in its zero-coordinate frame.

2.4.2. *Applying the marker geometry constraints*

Next, given all N detected feature observations in the image frame using the classification method described in Section 2.3 (potentially present on all visible tools as well as any potential false positives in the scene), we compute (1) an $N \times N$ distance matrix, where each distance matrix element (i,j)

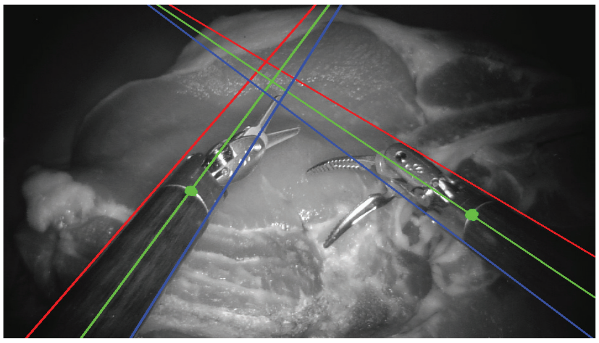


Fig. 6. By extracting the boundary lines of the shaft (red and blue lines), the mid-line axis (green lines), and then the intersection location between the tool's shaft and the clevis (green dot), we can add shaft observations along with the feature observations to the fusion stage of the tracking system.

specifies the 3D Euclidean distance between feature observation i and feature observation j, and (2) an $N \times N \times 3$ matrix of unit vectors, similar to those computed for the marker patterns using the kinematics estimates from the robot. To create the latter matrix, we compute the 3D unit direction vector between feature observation i and feature observation j.

We use these two matrices to reject any feature observations which do not adhere to one of the pre-processed marker distance and rotation configurations according to the PSMs. Using empirically determined distance (e.g. $\sim$ 3–5 mm) and orientation (e.g. $\sim$ 10°–20°) thresholds, we are able to determine which feature observation is associated with each PSM instrument, by rejecting those distance and unit vector observation entries which are inconsistent with an expected marker pattern, allowing only one assignment per feature class to each PSM instrument.

2.5. *Shaft extraction*

As mentioned earlier, it is not guaranteed that there are enough shaft pixels visible to compute valid cylinder estimates, and so we use stereo vision to estimate the distance of the tool tip from the camera. If the algorithm determines that the tools are situated far enough away from the camera that the shaft is sufficiently visible (based on a distance in the z-dimension along the optical axis from the camera which is empirically determined), we use the shaft likelihood mask (from Section 2.2) to collect pixels in the image (potentially) belonging to one of the two tools' shafts (module C in Figure 2). Assuming that each tool shaft is represented as a large, rectangular *blob* (see Figure 3, lower left, for an example), using connected components and 2D statistical measures (e.g. aspect ratios, total pixel areas) we eliminate those areas of the image which are not likely to be one of the tool shafts.

Next, we fit 2D boundary lines to each candidate shaft blob, as shown with the blue and red lines in Figure 6. Using projective geometry (Hartley and Zisserman, 2003) we fit a 3D cylinder to each pair of 2D lines, representing a single tool's shaft. Then, we locate the intersection point in the 2D image where the tool shaft meets the proximal clevis by moving along the cylinder axis mid-line from the boundary of the image and locating the largest jump in gray-scale luminance values, representing where the black shaft meets the metal clevis (the green circles in Figure 6). We then project a 3D ray from the image (e.g. using the camera's intrinsics parameters producing a 3D ray from the 2D pixel location) through this 2D shaft/clevis pixel to intersect with the 3D cylinder and localize on the surface of the tool's shaft. Finally, we project this 3D surface location onto the axis mid-line of the shaft, representing a rotationally invariant 3D feature on the shaft. This shaft feature is associated with its known marker location and is added to the fusion stage along with the feature classification detections (from Section 2.3).

The green lines in Figure 6 represent the axis mid-lines of each tool shaft's cylinder representation, which are obtained by taking the average mid-line between the extracted shaft boundaries for each tool. The intersection location between the black shaft and the metal clevis is used because this natural contrast will provide a very large jump in pixel value which is more reliably detected in most lighting situations. The green dots along the green lines represent these shaft/clevis intersection locations, and each is projected to the associated 3D cylinder axis mid-lines, each of which has a known, prior location on the shaft (and is invariant to the roll of the shaft) to be used as an additional input to the fusion stage, described next.

2.6. *Fusion and tracking*

Because we cannot ensure that the features that we chose to detect are always visible on any given frame, we combine the robot kinematics with the vision estimates to provide our final articulated pose across time (module E in Figure 2). The kinematics joint angles are typically available at a very high update rate, although they may not be very accurate due to the error accumulation at each joint.

For surgical robots like the da Vinci®, it is important to keep the instrument insertion point (also termed *remote center*) stationary. This means that one part of the robotic arm holding the instrument does not move during the surgery (i.e. it is *passive*). The error of the end effector pose comes from both the error in zero calibration of the potentiometers at the joints and the error in the kinematics chain due to the link lengths. These are mostly static because the errors from the passive setup joints (SJUs) have more influence on the overall error as they are further up in the kinematic chain and have longer link lengths than the active joints. Therefore, if we can solve for this constant error bias, we can apply this to the raw kinematics of the active joints and end up with fairly accurate overall joint angle estimates. This bias essentially amounts to a rigid-body pose adjustment at the stationary remote center. Although there is also error for the robotic arm holding the camera, when it does not move it is not necessary to include this in the error contributions. However, the errors observed in the arm holding the camera are similar to those which hold the tool manipulators (up to one inch of absolute error).

To perform these adjustments on-line, we use an extended Kalman filter (EKF). Before we describe the equations, we will first define some coordinate systems used in the notation below.

1. **True remote-center coordinate system (RCS)**: the true remote-center coordinate system corresponding to the coordinate system which is attached to the true remote center. It is fully determined by the SJUs of the robot and is the virtual base for all active joints.
2. **Kinematics remote-center coordinate system (KCS)**: this corresponds to the remote center pose as derived by the kinematics. It absorbs all of the errors in the SJUs into an error in its pose (6 DOFs).
3. **True instrument joint coordinate system (ICS)**: there is one coordinate system attached to each rigid segment of the robotic arm.

2.6.1. Process model The state variables contain the offset of the remote center pose. In particular, it is represented as the true pose of the remote center in the coordinate system of the remote center derived from the kinematics (the KCS). The true remote location in the KCS is $\mathbf{c}_R^K = [c_x, c_y, c_z]^{\mathrm{T}}$, and the rotation in the form of a unit quaternion vector is $\mathbf{q}_R^K = [q_0, q_x.q_y, q_z]^{\mathrm{T}}$. The vector to be estimated is $\mathbf{x}_t = [q_0, q_x.q_y, q_z, c_x, c_y, c_z]^{\mathrm{T}}$. We assume that it is either fixed or slowly changing, and therefore we model it as a constant process. The process noise can be tuned to find a balance of responsiveness and stability. Equation 11 below shows a simple static process model, where I_7 represents a 7×7 identity matrix and w_{t-1} represents the expected noise between time stamps of the measurements:

$$\mathbf{x}_t = I_7 \mathbf{x}_{t-1} + w_{t-1} \tag{11}$$

2.6.2. Observation models The observation model comes from our 3D point locations of our feature classes, transformed into the KCS. We need at least three non-collinear points for the system to be fully observable. The measurement vector is

$$\mathbf{y_3} = [x_1, y_1, z_1, \ldots, x_n, y_n, z_n]^{\mathrm{T}} \tag{12}$$

The observation function which transforms state variables to observations is not linear, and so we need to provide the following Jacobians:

$$\mathbf{J}_1 = \frac{\partial \mathbf{p}^K}{\partial \mathbf{q}_I^K} \tag{13}$$

$$\mathbf{J}_2 = \frac{\partial \mathbf{p}^K}{\partial \mathbf{c}_I^K} \tag{14}$$

where $\mathbf{p}^K$ is a 3D point location in the KCS, $\mathbf{q}_I^K$ is a unit quaternion rotation between the ICS and the KCS, and $\mathbf{c}_I^K$ is the remote center location in the KCS. For more details, we refer the interested reader to Zhao et al. (2009c). In practice, we found that the parameters of the EKF did not need to be finely tuned as long as we used reasonable estimates for the expected noise in the position and orientation of the remote center pose offset.

2.6.3. Handling outliers It is unlikely that any realistic solution to a computer vision problem does not contain outliers. We are mostly concerned with the image analysis as it is input to the fusion and tracking module (E in Figure 2). To deal with this, we add an initial RANSAC phase to gather a sufficient number of observations and perform a parametric fitting of the rigid transformation for the pose offset of the remote center. This is used to initialize the EKF and updates on-line as more temporal information is accumulated. We require a minimum of $\sim$ 30 total inliers for a sufficient solution to begin the filtering procedure. The rigid-body transformation offset is computed using the 3D correspondences between the class-labeled feature observations, done separately for each PSM after the PSM association stage described in Section 2.4, and the corresponding marker patterns after applying the forward kinematics estimates to the zero-coordinate frame locations for each tool. Because the remote center should not change over time, this pose offset will remain constant across the frames, and so by accumulating these point correspondences temporally, we are able to achieve a stable solution.

3. Experiments

We experimented on two types of datasets, both collected previously on a da Vinci® surgical robot: (1) porcine data (in vivo), and (2) pork data (ex vivo). The data which was used to test was specifically not included in the training collection procedure described in Section 2.3.1. After we collected and trained the seven feature classes using the $\sim$ 20,000 training samples with our best weighted RTs approach (from Section 2.3.5), we applied the PSM association and geometric constraints method from Section 2.4 and finally the fusion and tracking stage from Section 2.6. To best account for appearance variabilities of the landmarks, we made sure to include training samples of each landmark under many different conditions, including various scene illuminations, in-plane and out-of-plane rotations, and intermittent specularities. In this way, different variations of the appearance of each feature are captured in the classifier and invariance is achieved through the use of many different instances of each feature type.

Overall, we experimented on six different video sequences, totaling 6876 frames (i.e. 458 s worth of video). Each video frame had two tools visible at all times. Across

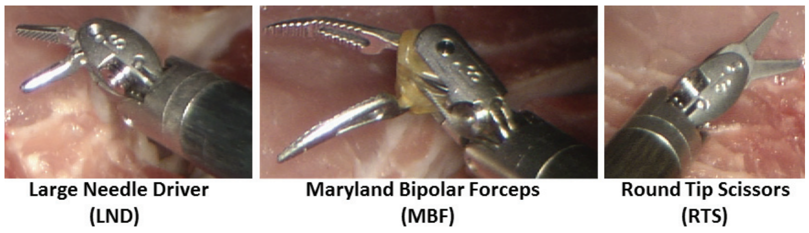


Fig. 7. Images of the three types of tool dealt with successfully in this paper. We train only on the LND (left), and are able to track on all three, including the MBF (middle) and RTS (right).

these video sequences, we experimented on three different types of da Vinci® tools, shown in Figure 7. To demonstrate the strength of our system, we trained only on the LND, shown on the left in Figure 7, and tested on that same LND tool in addition to the MBF (middle) and RTS (right). The method works on these other tools because there are many shared parts across the tools, including the pins used to hold the clevis together and the **IS** logo in the center of the clevis. Even though the overall appearance of each tool is quite different, our results show that the method extends very well to different tools given that the lower-level features are consistent. However, if newer tools are introduced which do not share these parts in common, more training data and feature classes must be considered and included in training the classifier discussed in Section 2.3.

We show 10 sample results in Figure 8 from various test sequences. Rows 1–4 show ex-vivo pork results with different combinations of the LND, MBF, and RTS tools. Row 5 shows a porcine in-vivo sequence with an MBF on the left and an LND on the right. In Row 4 on the right side, one tool is completely occluding the other tool's tip, however, the EKF from the fusion stage assists in predicting the correct configuration. In general, this is the typical behavior observed for partially visible or occluded tools *only if* the features were previously detected and accumulated over time. The reason for this is that the remote center for the robot arm is reasonably static, and so once the pose offset at the insertion point is accurately corrected, we can use the EKF to predict forward the remaining active joints, even when the features are not completely visible. The feature detections are important for reducing the drift of the arm at the remote center, however, for small periods of time using joint predictions from the EKF suffices for accurate overall tracking, and is a strength of our approach since it is inevitable that tools periodically exit the video frame. For each, the red lines portray the *raw* kinematics estimates as given by the robot. The blue lines show the *corrected* kinematics after running our detection and tracking system. We show full video sequences for each of these as follows:

- Row 1 ('Seq. 1'), MBF (left) and LND (right): http://www.youtube.com/watch?v=EWWQd-3zIT4;
- Row 2 ('Seq. 2'), RTS (left) and MBF (right): http://www.youtube.com/watch?v=fT1wILqpY6w;
- Row 3 ('Seq. 3'), LND (left) and RTS (right): http://www.youtube.com/watch?v=DD6ucZv5l2o;
- Row 4 ('Seq. 4'), MBF (left) and MBF (right): http://www.youtube.com/watch?v=aGXR3BytGRs;
- Row 5 ('Seq. 5'), MBF (left) and LND (right): http://www.youtube.com/watch?v=cNV12l_559g.

One additional video result ('Seq. 6') is shown at http://www.youtube.com/watch?v=TKiFQ3fKouM. In these sequences, again the red lines represent the raw kinematics estimates from the robot, projected onto the image frames. Notice the significant errors, where in some images the estimates are not visible at all, motivating the need for the algorithms presented in this paper. The blue lines represent the corrected kinematics resulting from our tracking system. A visual inspection yields a fairly accurate correction of the kinematics overlaid on the tools.

Because joint-level ground truth for articulated tools is very difficult to collect accurately and on a large dataset, we evaluated the accuracy of our tracking system in the 2D image space. The left image in Figure 9 describes our evaluation scheme for our kinematics estimates. The dotted blue lines define an acceptable boundary for the camera-projection of the kinematics, where the green line is a perfect result. The right image in Figure 9 shows an example of an incorrect track on the rightmost tool. Using this scheme, we manually inspect each frame of the test sequences, resulting in a 97.81% accuracy rate over the entire dataset.

Table 1 shows a more detailed breakdown of our evaluation. Overall, we tested against six sequences, including both ex-vivo and in-vivo environments and all had two tools in the scene. The table shows the test sequence name in the first (leftmost) column, the number of tracks labeled as correct in the second column, the total possible number of detections in that sequence in the third column, and the final percentage correct in the last (rightmost) column. Note that in any given frame, there may be one or two tools visible, and this is how we compute the numbers in the third column for the total potential number of tracks in that sequence. Finally, the last row shows the total number of correct tracks detected as 13,315 out of a total possible of 13,613, yielding our final accuracy of 97.81% correct. Also note that the accuracy was very similar across the sequences, showing the consistency of the algorithm. Although the accuracy was evaluated in the 2D image space, we note that this

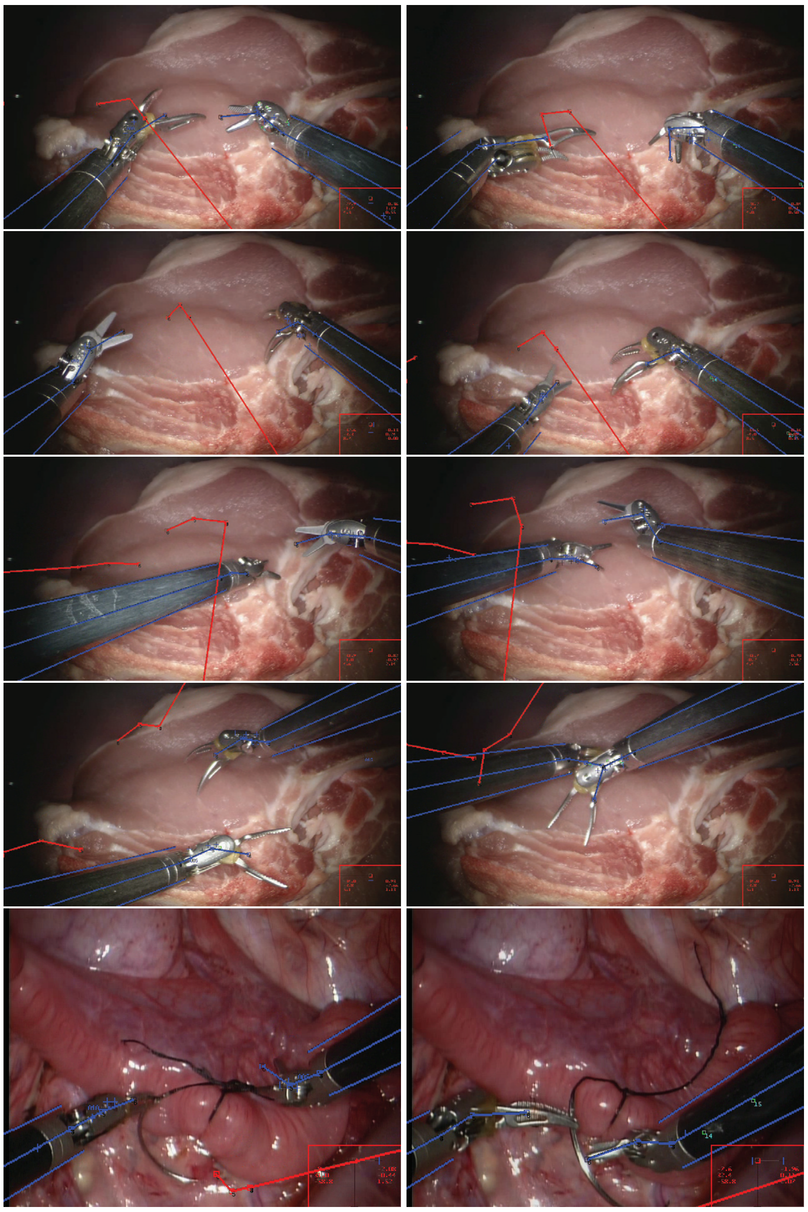


Fig. 8. Ten sample results from various test sequences. Rows 1–4 show different combinations of the three tools in Figure 7 on the ex-vivo pork sequence. The last row shows a porcine in-vivo sequence. For each, the red lines show the *raw* kinematics estimates from the robot and the blue lines show the *corrected* kinematics after running our tracking algorithm. Row 1: MBF (left) and LND (right); Row 2: RTS (left) and MBF (right); Row 3: LND (left) and RTS (right); Row 4: MBF (left) and MBF (right); Row 5: MBF (left) and LND (right). The right side of Row 4 shows one tool occluding the other, however, the EKF helps to predict through that.

does not completely represent the overall 3D accuracy as errors in depth may not be reflected in the perspective image projections.

The full tracking system runs at approximately 1.0–1.5 s/frame using full-sized stereo images (960 × 540 pixels). The stereo matching, PSM association, and fusion/EKF updates are negligible compared to the feature classification and detection, which takes up most of the processing time. This is dependent on the following factors: number of trees in Λ, depth of each tree γ_i, number of features used in the region covariance descriptor $\mathbf{C_R}$ (we use 11, but fewer could be used), and the quality of the initial segmentation providing the mask prior. However, by half-sizing the images we can achieve a faster frame-rate (0.6–0.8 s/frame, an example of which is shown in Seq. 5) while achieving similar accuracy. Also, because we are solving for a remote-center bias offset which remains constant over time, we can afford to process the frames at a slower rate without affecting the overall accuracy of the tracking system. Finally, many stages of the classification are parallelizable, and we are currently looking at implementing both the covariance descriptor and RTs on a GPU processor. Preliminary results

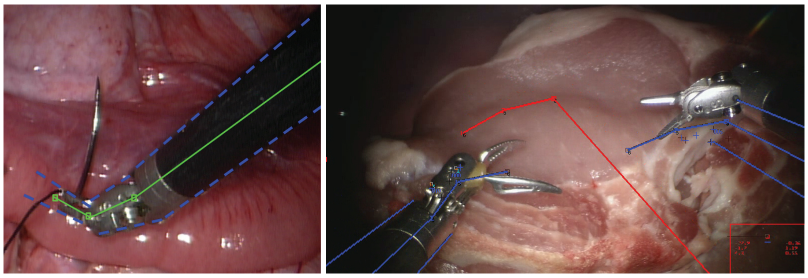


Fig. 9. Left: to evaluate our kinematics estimates, we evaluate in the image space because of the difficulty in collecting ground truth. The projected overlays must fall within the boundaries labeled as dotted blue lines here, and the green is a perfect detection. Right: an example of an *incorrect* track on the rightmost tool.

Table 1. Tracking accuracy breakdowns.

Sequence	# Correct	Potential	% Correct
Seq. 1	1890	1946	97.12%
Seq. 2	2114	2182	96.88%
Seq. 3	1447	1476	98.04%
Seq. 4	1611	1648	97.75%
Seq. 5	4376	4431	98.76%
Seq. 6	1877	1930	97.25%
TOTAL	13,315	13,613	97.81%

on the covariance processing reduces the processing time of the feature tensors (equation (1)) from $\sim$ 700 ms to $\sim$ 100 ms, and we believe we can reduce this further. We save for future work the GPU parallelization of the descriptor and classification procedures.

4. Discussions

4.1. Descriptor window size

There are many important choices to be made when implementing this tracking system. One such decision is the size of the window to use when extracting covariance descriptors for classification throughout the image. The reason is that, during training, we use the best encompassing bounding box around each feature, and the descriptors are well tuned to representing the entire feature. When applying the classifier, if the window is too small or too large, the descriptors will not capture the features well. To alleviate this, we use prior knowledge of the 3D sizes of the features to guide computation of the optimal window size. Using the stereo vision approach which determines if the shaft is visible enough to extract (from Section 2.2) and estimating that the features are $\sim 3 \times 3$ mm in size, we can automatically determine the optimal window size in the image *dynamically* on each frame. To further reduce errors, at every pixel location that we evaluate, we extract a bounding box which is both full- and half-sized according to this automatically determined window size to account for the smaller features (e.g. the pins). This improves the overall feature detection system.

4.2. Kinematics latency

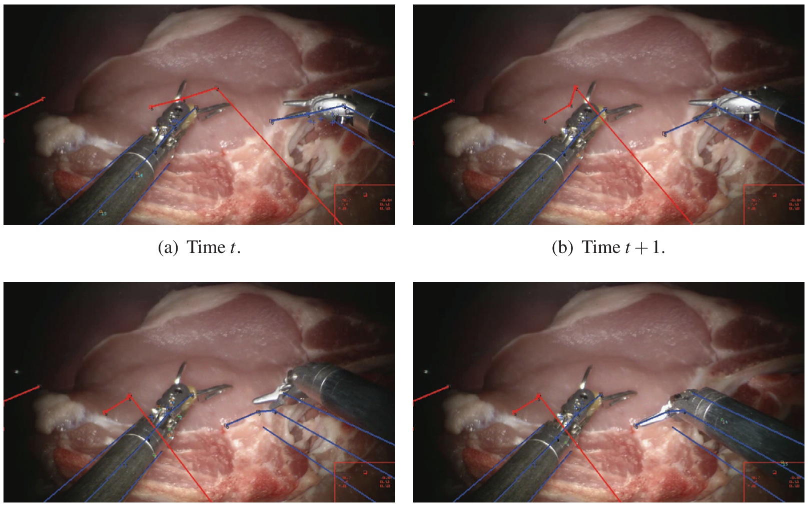
Upon further inspection of the errors encountered during evaluation on the test sequences, we found that most of the incorrect fixed/tracked kinematic configurations are due to a latency in the raw kinematics which causes the video and raw kinematics to be out of sync from time to time. This situation is shown more precisely in Figure 10. We determined this by noticing that, for the individual frames which had incorrect projections (according to our scheme described in Section 3), the result would jump immediately to a correct configuration instead of getting lost completely, and the previous incorrect projection was in the location and configuration that the upcoming projection would eventually reach. Therefore, by logging the test data more precisely so that the video and kinematics are more in sync with each other, we would expect our accuracy to increase even further. However, in practice on a live system this kinematic latency does not exist, and future in-vivo live experiments should demonstrate this.

4.3. Other error sources

In addition to the errors due to the kinematics/video synchronization, the remaining errors that we observed were due to incorrect feature assignments. This error presented in one of two forms: either assigning a particular feature to the wrong tool (given that more than one is visible), or accepting the wrong location for a particular feature class on the correct tool. For these types of errors, we use geometric constraints of distances and orientations between the feature candidates to reject geometrically inconsistent detections. However, occasionally too many incorrect detections are assigned to the wrong tool, causing the tool's pose to be off. Over time, as more features are correctly assigned, this error tends to correct itself.

4.4. Hybrid approach

Finally, we wish to mention that the majority of tool tracking approaches in the literature work by estimating the cylinder of the shaft which is visible in the scene (Doignon et al., 2006; Voros et al., 2007). However, as we previously


(a) Time t. (b) Time $t+1$.

(c) Time $t+2$. (d) Time $t+3$.

Fig. 10. Example of kinematic latency (right tool): often the kinematics and video get out of sync with each other. Most of our errors are due to this fact, manifesting in the situation shown here. In (a), both tools are tracked well. Then, in (b) and (c), the kinematics and video become out of sync and the right tool becomes inaccurately tracked. However, in (d), the tools are tracked successfully again. The blue configuration in (c), which is essentially the same as the correct one immediately following in (d), suggests this latency is the source of our errors. These four frames are consecutive to each other in order.

discussed, surgeons tend to work quite zoomed in, making this cylinder-fitting procedure very difficult, if not impossible, due to the limited number of visible shaft pixels. The remaining minority approaches work by analyzing the tip of the tool using features (Burschka et al., 2004; Reiter and Allen, 2010), however, these will fail when the tool tip is too far away to be seen well by the camera. Our approach is advantageous in that it dynamically decides which of these two approaches is optimal at any given time, and often uses both simultaneously to best the track the tool over longer periods of time. Also, by using the pixel-labeling method described in Section 2.2, we are able to tell more accurately when parts of the tool are occluded. For example, if the metal tool tip is occluded then the pixel labeling will not label the incorrect pixels from the occluder as metal, and we will avoid fewer false positives, and similarly for the shaft.

4.5. *Other tooling*

Although the contents of this paper have been exclusively proven on the da Vinci® robot, the method is generic enough to be applicable to other types of laparoscopic tool, either manually or robotically controlled. The algorithm requires the presence of some kinds of fiducials, either naturally occurring or manually placed. The advantages shown were in the ability to detect very small features robustly, and so even structures which are naturally occurring such as pins and miniature ridges within the tool's tip can be used as features. This decreases the reliance on carefully placed markers, which are challenging to manufacture accurately. Many existing surgical tools do in fact have such features, and so our approach could be applied to these tools as well.

5. Conclusions

This paper has presented a tool detection and tracking framework which is capable of tracking multiple types of tools and multiple tools simultaneously. The algorithm was demonstrated on the da Vinci® surgical robot, however, it may be extended to other types of surgical robot. We showed high accuracy and long tracking times across different kinds of environment (ex vivo and in vivo). By learning low-level features using a multi-class classifier, we showed how different degrees of visibility for each feature can be overcome. We also showed that a hybrid approach of using both the shaft and features on the tool tip is advantageous over either of these methods alone. Using knowledge of the distance of the tool we can dynamically adapt to different levels of information into a common fusion framework. Finally, by fusing vision and kinematics, we can account for missed observations over time.

Funding

This work was funded by Intuitive Surgical, Inc., CA, USA.

References

Allan M, Ourselin S, Thompson S, et al. (2013) Toward detection and localization of instruments in minimally invasive surgery. *IEEE Transactions on Biomedical Engineering* 60(4): 1050–1058.

Bosch A, Zisserman A and Munoz X (2007) Representing shape with a spatial pyramid kernel. In: *ACM International conference on image and video retrieval*.

Burschka D, Corso JJ, Dewan M, et al. (2004) Navigating inner space: 3-D assistance for minimally invasive surgery. In:

IEEE/RSJ International conference on intelligent robots and systems.

Cortes C and Vapnik VN (1995) Support-vector networks. *Machine Learning* 20: 273–297.

Dalal N and Triggs B (2005) Histograms of oriented gradients for human detection. In: *IEEE conference on computer vision and pattern recognition*.

Doignon C, Graebling P and de Mathelin M (2005) Real-time segmentation of surgical instruments inside the abdominal cavity using a joint hue saturation color feature. *Real-Time Imaging* 11(5–6): 429–442.

Doignon C, Nageotte F and de Mathelin M (2004) Detection of grey regions in color images: Application to the segmentation of a surgical instrument in robotized laparoscopy. In: *IEEE/RSJ International conference on intelligent robots and systems*.

Doignon C, Nageotte F and de Mathelin M (2006) Segmentation and guidance of multiple rigid objects for intra-operative endoscopic vision. In: *European conference on computer vision*, pp. 314–327.

Duda RO, Hart PE and Stork DG (2001) *Pattern Classification*. 2nd edn. New York: Wiley.

Groeger M, Arbter K and Hirzinger G (2008) Motion tracking for minimally invasive robotic surgery. In: Bozovic V (ed.) *Medical Robotics*. Vienna, Austria: I-Tech Education and Publishing, pp. 117–148.

Hartley R and Zisserman A (2003) *Multiple View Geometry in Computer Vision*. 2nd edn. New York, NY: Cambridge University Press.

Intuitive Surgical, Inc. (1995) Homepage at: http://www. intuitivesurgical.com/.

Krupa A, Gangloff J, Doignon C, et al. (2003) Autonomous 3-D positioning of surgical instruments in robotized laparoscopic surgery using visual servoing. *IEEE Transactions on Robotics and Automation, Special Issue on Medical Robotics* 19(5): 842–853.

Lee C, Wang YF, Uecker D, et al. (1994) Image analysis for automated tracking in robot-assisted endoscopic surgery. In: *Proceedings of the 12th International conference on pattern recognition*.

Lepetit V and Fua P (2006) Keypoint recognition using randomized trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28(9): 1465–1479.

Leven J, Burschka D, Kumar R, et al. (2005) DaVinci canvas: A telerobotic surgical system with integrated, robot-assisted, laparoscopic ultrasound capability. In: *International conference on medical image computing and computer assisted intervention*, pp. 811–818.

Lowe D (2004) Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision* 60(2): 91–110.

Lucas B and Kanade T (1981) An iterative image registration technique with an application to stereo vision. In: *International joint conference on artificial intelligence*, pp. 674–679.

Mack MJ (2001) Minimally invasive and robotic surgery. *The Journal of the American Medical Association* 285: 568–572.

Malpani A, Vagvolgyi B and Kumar R (2011) Kinematics based safety operation mechanism for robotic surgery extending the JHU SAW framework. *The MIDAS Journal – Systems and Architectures for Computer Assisted Interventions*

Pennec X, Fillard P and Ayache N (2006) A Riemannian framework for tensor computing. *International Journal of Computer Vision* 66(1): 41–66.

Pezzementi Z, Voros S and Hager G (2009) Articulated object tracking by rendering consistent appearance parts. In: *IEEE International conference on robotics and automation*.

Reiter A, Allen PK and Zhao T (2012a) Feature classification for tracking articulated surgical tools. In: *International conference on medical image computing and computer assisted intervention*, pp. 592–600.

Reiter A, Allen PK and Zhao T (2012b) Learning features on robotic surgical tools. In: *Workshop on medical computer vision, IEEE conference on computer vision and pattern recognition*.

Reiter A, Allen PK and Zhao T (2012c) Marker-less articulated surgical tool detection. In: *Computer assisted radiology and surgery*.

Reiter A and Allen PK (2010) An online approach to in-vivo tracking using synergistic features. In: *IEEE/RSJ International conference on Intelligent robots and systems*.

Tuzel O, Porikli F and Meer P (2006) Region covariance: A fast descriptor for detection and classification. In: *European conference on computer vision*.

Tuzel O, Porikli F and Meer P (2007) Human detection via classification on Riemannian manifolds. In: *IEEE conference on computer vision and pattern recognition*.

Viola PA and Jones MJ (2004) Robust real-time face detection. *International Journal of Computer Vision* 57(2): 137–154.

Voros S, Long J and Cinquin P (2007) Automatic detection of instruments in laparoscopic images: A first step towards high-level command of robotic endoscopic holders. *The International Journal of Robotics Research* 26(11–12): 1173–1190.

Wei GQ, Arbter K and Hirzinger G (1997a) Automatic tracking of laparoscopic instruments by color coding. In: *Proceedings of the first joint conference on computer vision, virtual reality and robotics in medicine and medical robotics and computer-assisted surgery*, pp. 357–366.

Wei GQ, Arbter K and Hirzinger G (1997b) Real-time visual servoing for laparoscopic surgery. *IEEE Engineering in Medicine and Biology Magazine* 16(1): 40–45.

Wolf R, Duchateau J, Cinquin P, et al. (2011) 3D tracking of laparoscopic instruments using statistical and geometric modeling. In: *International conference on medical image computing and computer assisted intervention*, pp. 203–210.

Zhang X and Payandeh S (2002) Application of visual tracking for robot-assisted laparoscopic surgery. *Journal of Robotic Systems* 19(7): 315–328.

Zhang Z (2000) A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22(11): 1330–1334.

Zhao T, Zhao W and Nowlin W (2009a) *Configuration marker design and detection for instrument tracking.* US Patent US 2010/0168763 A1.

Zhao T, Zhao W, Halabe D, et al. (2009b) *Fiducial marker design and detection for locating surgical instrument in images.* US Patent US 2010/0168562 A1.

Zhao T, Zhao W, Hoffman BD, et al. (2009c) *Efficient vision and kinematic data fusion for robotic surgical instruments and other applications.* US Patent US 2010/0331855 A1.