Depth Cameras, Machine Learning, and Temporal Tracking: Implementing an On-The-Fly Tracker

Henri Prudhomme, Rich Deluca, *IMT*, and Tony Dowson, *IMT*

***Abstract*—A team at Stryker is working to develop a machine vision toolkit for use in future products. The current algorithm has a critical dependency on owning a synthetic 3d representation of any object that is to be tracked. This research investigates a proposal by Tan to use online learning to learn to track any unknown object on the fly. Methods for registration, filtering, segmentation, and parameterization are covered in this paper.**

***Index Terms*—Temporal Tracking, Homogeneous Transform, Online Learning, Machine Learning, Random Forest Regression.**

I. Introduction

STRYKER is working to develop a machine vision toolkit for use in its future products. To accomplish this, the company is currently researching the work of Dr. David Joseph Tan on 6-DoF pose estimation and temporal tracking. Specifically, Stryker is investigating two of his more recent papers *"Learn To Track: From Images To 3D Data"* [2] and *"Looking Beyond The Simple Scenarios: Combining Learners And Optimisers In 3D Temporal Tracking"* [3].

Before discussing any of the implementations of such an algorithm, it is necessary to discuss some of the mathematics integral to tracking.

Tracking involves knowing both a position and orientation of a body relative to a reference frame. See Figure 1. The construct used here is a homogeneous transform (1). This transform is also known as a 6 DoF Euclidean Transform. This transform is a 4x4 matrix composed of both a 3x3 3d rotation matrix (2), as well as a 3d position vector (3). The rotation matrix is the product of the axial rotation matrices (4, 5, 6). The order in which these matrices are multiplied is called a sequence. We used a XYZ sequence as seen in equation (2).

$$T = \begin{bmatrix} \tilde{R}_{3x3} & \tilde{t} \\ 0^{\top} & 1 \end{bmatrix} \tag{1}$$

$$\tilde{R}_{3x3} = R_z R_y R_x \tag{2}$$

$$\tilde{t} = [X, Y, Z] \tag{3}$$

$$R_z = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

This project was sponsored by Stryker IMT
Henri Prudhomme is with Stryker IMT as a Software Engineer
Rich DeLuca is with Stryker IMT as a Senior Principal Engineer.
Tony Dowson is with Stryker IMT as a Senior Staff Engineer.
Manuscript submitted May 16, 2021

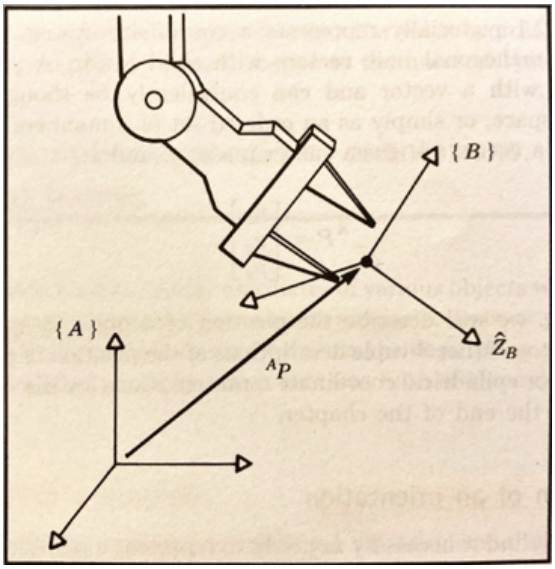


Fig. 1. Body Frame B relative to a Reference Frame A (from *Introduction to Robotics* [1])

$$R_y = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \tag{5}$$

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix} \tag{6}$$

The most important feature of a homogeneous transform is its ability to multiply with other transforms to update a position. For example, given a body frame (7) located at [1, 0, 1], we can rotate it about its z axis by applying the transform ΔT (8).

$$T = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

$$\Delta T = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

$$\Delta T * T_t = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

Given the nature of homogeneous transforms, a method for tracking has been derived which depends on this behavior.

Temporal Tracking is a specific approach for tracking an object which involves updating a known pose from frame to frame. The algorithm is as follows:

Algorithm 1 Temporal Tracking Algorithm

$T_{current} \leftarrow T_{initial}$ ▷ Start from a known transform
while true **do**
  $\Delta T \leftarrow Predict(...)$ ▷ Find ΔT
  $T_{current} \leftarrow \Delta T * T_{current}$ ▷ Update transform
end while

What makes this approach unique is that registration of the object in the scene happens only one time. On the first iteration, the transform must be configured to the initial pose of the object during the tracking sequence. During subsequent iterations, only the ΔT between the frames must be determined. This speeds up the tracking speed considerably compared to other methods such as Iterative Closest Point in which registration must occur every iteration.

II. Related Work

Tan's work [2, 3] is focused on developing and managing a collection of random forest regression models which predict the ΔT for updating a pose. He has been successful and able to accomplish a tracking speed of 2ms per iteration with tracking precision to just a millimeter or so of error. Stryker has obtained similar results implementing a model-based tracker.

However, a key dependency of Tan's approach is that he uses synthetic meshes to train his model before tracking. Any object for which a synthetic 3d model cannot not be obtained cannot be tracked by Tan's method.

Tan suggested an approach which could solve this difficulty. He proposed to *"From one frame to the next, incrementally add new trees to the forest from different object viewpoints"*. This approach is called online learning. It is a method of learning to track an object on-the-fly. By implementing such a method, we can enable our tracker to adaptively learn to track objects that it has not encountered before.

III. Method Overview

Tan's proposal for the online learning algorithm was the following:

> *In particular, from one frame to the next, to incrementally add new trees to the forest from different object viewpoint. To achieve this goal, the online learning is initialized by defining the object to learn in the first frame and a 3D bounding box that encloses the object. It follows that the centroid of the box is the origin of the object and the object transformation of the initial frame T^ 0 is the translation from the camera center to the centroid. The bounding box defines the constraints of the object in 3D space and segments the object for learning.*

We have broken this proposal into 4 main steps: predict, update, segment, and learn. The process is shown in Figure 2.

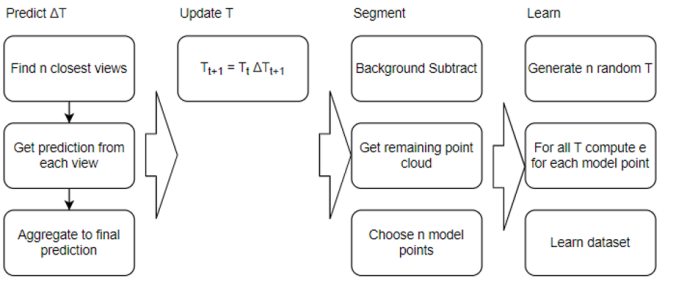


Fig. 2. The process loop during online tracking

A. Predict: Find Closest Views

Predictors which have been trained from viewpoints similar to the current transform provide better pose estimates. To select these predictors, we apply the following algorithm.

Algorithm 2 FindClosestViewsAngular

for each *view* **do**
  $collection \leftarrow N_c \cdot N_v$ ▷ Record magnitude
end for
$sort(collection)$ ▷ Order from highest to lowest
$closestViews \leftarrow collection[1:n]$ ▷ Get n closest views

1) For each view, compute the scalar given by the product of the camera normal with the view normal. $U = N_c \cdot N_v$
2) Sort the set of scalars by magnitude. The resulting set is ordered with respect to which views are angularly closest to the current transform.
3) Select the n closest views

Figure 3 demonstrates the selection of the 35 closest views in the collection of trained views to the current transform (seen in orange).

After using the predictors to get a ΔT estimate, we aggregate the predictions using one of a variety of methods. A simple method we can use is to take the mean estimate of the delta transforms.

B. Update T

When updating the pose, with $T_{t+1} = \Delta T * T_t$. It is important to keep track of which side ΔT is applied to T. Matrix multiplication is not commutative. Whether a right or left multiply is needed depends on the implementation.

C. Segment Image

Input to the tracker is a depth image from an Intel RealSense D435 camera. A sample image from the camera is shown in Figure 4.

In order to better manage what a forest learns on, we segment the object from the image. To accomplish this we use simple background subtraction. The algorithm for background subtraction is the following:

1) Take a depth snapshot of the scene.

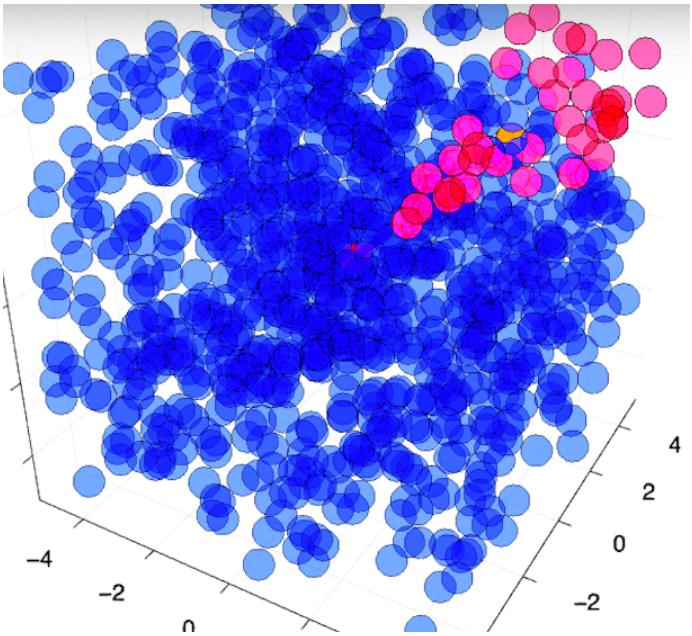


Fig. 3. Trained views are blue and red, selected views are red, and the current transform is orange

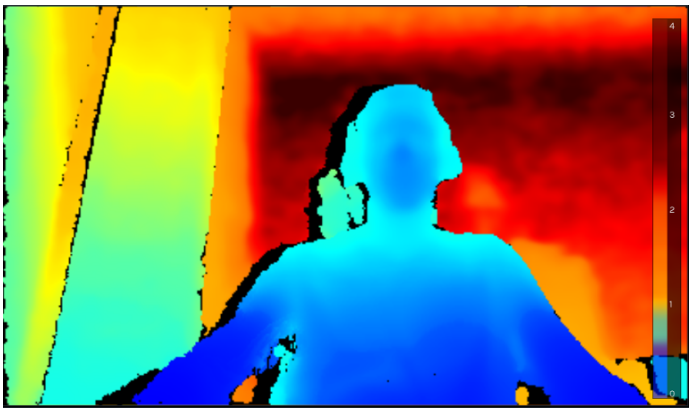


Fig. 4. A sample depth image from an Intel RealSense D435

2) When tracking subtract the scene from the current image and only keep values where the negative difference is below a threshold value.

Figure 5 illustrates this process and the result.

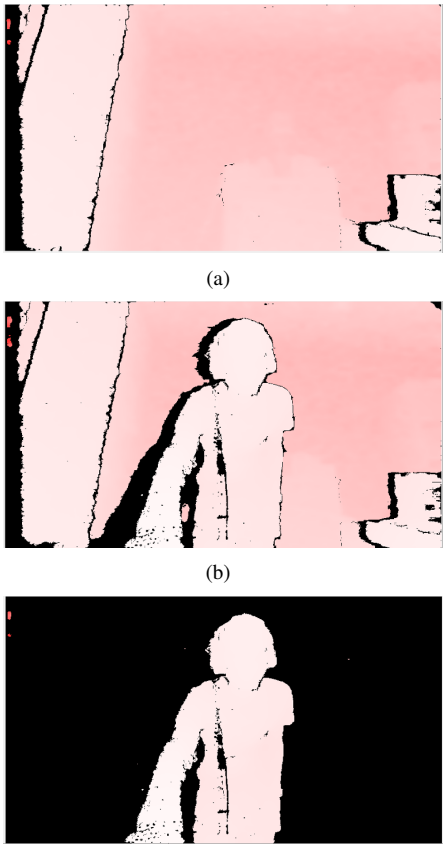
From this image we apply simple object detection to obtain an estimate for the object initial pose as well as to select model points for training. See Figure 6.

By using the bounding box to segment the image, we can project the depth map to get a 3d point cloud. We assume a pinhole camera configuration and project the depth values to 3d space using the camera intrinsic parameters. See Figure 7.

We use the projection formula (10) to obtain a 3d point from the depth value of a pixel.

$$\begin{cases} X = d(x,y) * (x - x_{principalpoint})/x_{focallength} \\ Y = d(x,y) * (y - y_{principalpoint})/y_{focallength} \\ Z = d(x,y) \end{cases} \tag{10}$$

Figure 8 provides an example of a possible point cloud.



(a)

(b)

(c)

Fig. 5. (a) Scene before tracking (b) Frame during tracking (c) Image after background subtraction

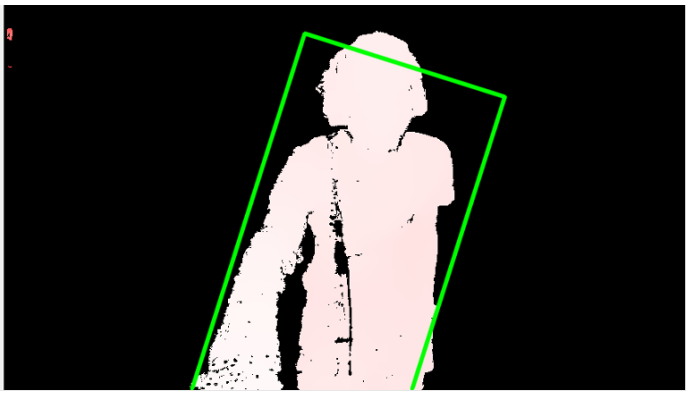


Fig. 6. A background subtracted image with an overlay of the detected object bounding box

This completes the process of segmenting an image and obtaining model points to learn from.

D. Learn: Error

The forests that Tan develops, produce a ΔT estimate from an input error vector. Error is a scalar which describes the displacement of model points relative to the transform between the predicting forest and the object, and is given by equation (11)

$$\epsilon_j^v(T;D) = N_v \cdot (T^{-1}\mathcal{D}(x_j) - X_j) \tag{11}$$

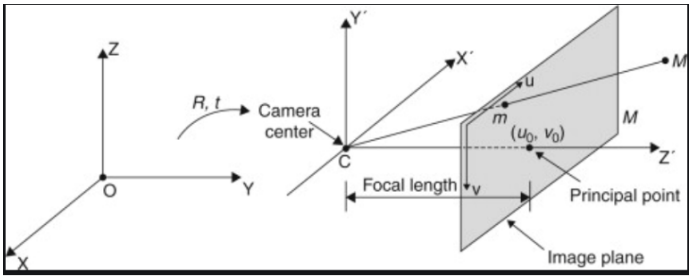


Fig. 7. A labelled diagram illustrating where camera intrinsic parameters originate

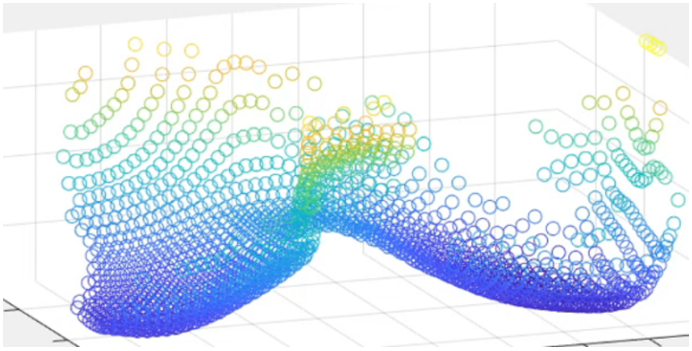


Fig. 8. 3d point cloud of the patellar surface of a femur.

where $\mathcal{D}$ applies a backprojection of the model points X_j through the camera. Error can be visualized as the displacement vector between the current position and the backprojected scene points $T^{-1}\mathcal{D}(x_j)$. See Figure 9. See Tan's work for a description of projective point correspondence [2, 3].

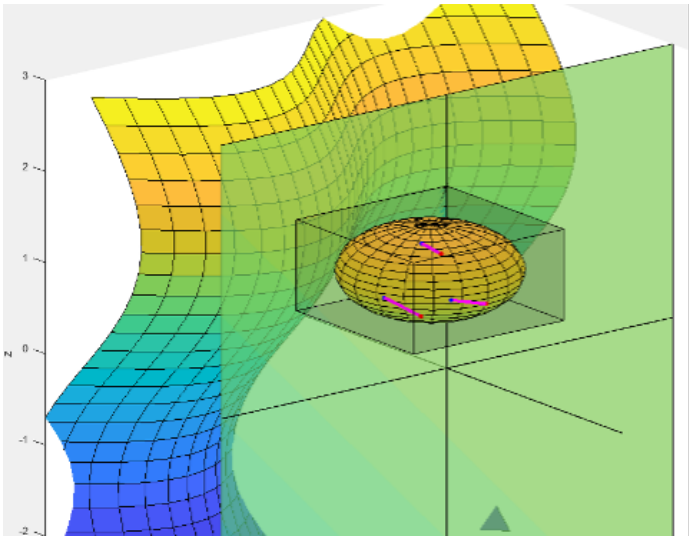


Fig. 9. Blue model points, Red scene points, and magenta displacement vectors between the model points and the backprojected scene points.

The decision forests learn the relation between displacements of model points and the transforms that produce said displacements.

E. Learn: Online Learning

Per frame, we take model points from the scene, apply n random transformations to the model points, compute an error vector using (11), and finally package the transform and error vector as a single row of training data. Because the actual learning algorithm was not a focus of this research, we omit those details.

After training a random forest with the generated data, a new view is created and saved for future predictions.

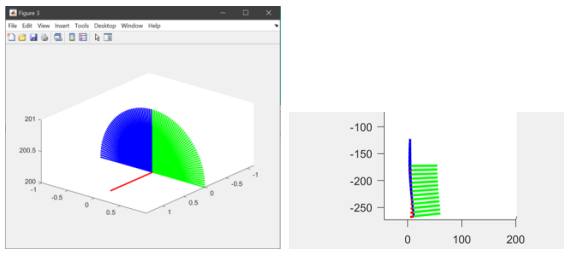
This process of adding news forests per frame is the essence of the online learning algorithm.

IV. Experiments

We have not yet performed experiments on the tracker. A working implementation must be completed before optimal parameterization can be tested.

V. Conclusion

The current version of the online tracker fails to operate as expected. Measured output shows severe drift indicating that some portion of the algorithm has not been implemented correctly. Figure 10 shows how the expected output differs from the actual.



(a) (b)

Fig. 10. (a) Theoretical transform movement (b) Measured transform movement

In order to continue development, plans are in place to test the API methods in a model based environment to ensure that the basic functionality is properly implemented. There are also plans to test parameters such as view density, number of initial forests, and segmentation parameters for the purpose of tuning.

References

[1] J. Craig, *Introduction to Robotics: Mechanics and Control*, Addison-Wesley, 1986.

[2] D.J. Tan. *Learn to Track From Images to 3d Data*. [Unpublished doctoral dissertation]. University of Munich, 2015.

[3] D.J. Tan, N. Navab and F. Tombar, *Looking Beyond the Simple Scenarios: Combining Learners and Optimizers in 3d Temporal Tracking*, in IEEE Transactions on Visualization and Computer Graphics, vol. 23, no. 11, pp. 2399-2409, Nov. 2017, doi: 10.1109/TVCG.2017.2734539.