Particle Filter based RGB-D Tracking for Mobile Robots

Arnav Dhamija Mihir Parmar

May 22, 2020

Abstract

In this report, we revisit the problem of 3D object tracking using an RGB-D camera. We present a novel method of tracking an object's translation and rotation relative to a camera using an RGB-D camera such as the Microsoft Kinect or Intel Realsense D435i. Our approach uses a particle filter with correlation computed using 2D feature descriptors. This approach can handle the cases of estimating the position of objects out of the camera's depth range, behind obstacles, and out of the camera's frame. This approach can be extended for active tracking objects on mobile robots.

1 Motivation

Mobile robots, such as Unmanned Aerial Vehicles (UAVs), have recently seen a surge in popularity for consumer applications such as photography and videography. Companies such as DJI and Skydio offer quadcopters which are advertised with automatic object tracking capabilities using onboard sensors and cameras. These drones typically use monocular RGB cameras for tracking an object at a fixed distance. These approaches have limited accuracy as the tracking algorithm needs to estimate the distance of the object from the drone. Deep learning models, such as OpenPose [1], can estimate the object's size and position given an RGB image, but these detectors need to be trained separately for different classes of objects, limiting the flexibility of the tracking approach.

Hence, we wanted to explore a more general solution for this problem using an RGB-D camera. An RGB-D camera can sense the depth for each pixel in the camera's image plane. This can be used for reliably tracking an object's relative position to the camera. However, most commercially available depth cameras have a limited 5m range for depth sensing. One possible method to track objects beyond the depth range of the camera is to store a 3D model of the object of interest with the relative 3D positions of its 2D feature descriptors. By observing the positions of these 2D feature descriptors in the subsequent frames, we can then apply the Perspective-Three-Point (P3P) algorithm for calculating the relative position of the object. However, P3P approaches [2] are not robust to the outliers which may occur due to errors in keypoint matching. P3P may also return multiple possible solutions. The best solution must then be determined by using heuristics, which may not always be reliable. The P3P approach will not also be able to estimate the position of the object due to occlusions or poor keypoint matching.

To this end, we propose a real-time method which can estimate the position of an object in the three cases of tracking an object with an RGB-D camera, namely 1) the object is both visible in the RGB image and detected by the depth camera 2) the object is only visible in the RGB image and 3) the object is no longer in view of both the RGB camera and the depth map. In our approach, we implemented a particle filter to track an arbitrary object by calculating the relative positions of its features descriptors to the center of the object when the object is in range of the depth sensor. For the purpose of this project, we detect the 2D bounding box of the object in the RGB camera using a real-time 2D tracker such as MobileNetSSD or CSRT as we wish to focus on the 3D tracking aspect of this problem.

2 Literature Review

We considered several different approaches for learning how object tracking is currently implemented on mobile robots such as drones. Most of the current literature deals with 2D object tracking using conventional RGB cameras. Bartak [3] describes an active tracking approach for the Parrot AR.Drone using a tracking-learning-detection (TLD) approach [4]. This approach uses a PID controller to track the object, with the error being the change in scale of the detected bounding box from its initial detection. Our initial approach was based around deploying our algorithm on the DJI Tello drone. However, on testing OpenCV's implementation of the TLD tracker, we found that the tracker performed poorly on re-detecting the object after occlusion. We also found the DJI Tello's SDK too limited for our requirements.

Wang [5] presents an approach similar to the problem we are trying to solve in the context of feature based visual SLAM methods. This paper uses EPnP [2] for finding the camera pose based on features points, using the pixel locations of the points and their corresponding co-ordinates in the camera frame obtained from the depth camera. However, this approach would not be suitable for tracking if the object happens to go out of the depth sensor's range. We also considered filtering based approaches such as the "Simple, Online, and Realtime Tracking" (SORT) [6] which uses a Kalman filter for predicting the obstacles location between bounding box detections and uses Intersection Over Union (IOU) for calculating bounding box correspondences between frames. We found that SORT was not able to successfully associate objects correctly after occlusion. However, the ideas of using filtering for object detection inspired us to look into using a particle filter for implementing our approach.

3 Approach

3.1 Particle Filter Approach

The key idea of particle filters is to represent and maintain the posterior density function by a set of random samples with associated weights and to compute the state estimate from those samples and those weights. For our problem of object tracking, we use a particle filter to represent the belief distribution of the location of the center of the tracked object. The particles are assigned weights on the basis of the measurement obtained from the RGB-D camera and then using the best particle and the current camera pose, we estimate the relative pose using methods such as orthogonal Procrustes analysis for 3D correspondences. For the sake of brevity, this procedure is not discussed in this report, but the reader may refer to [7] for a discussion of this approach.

The aim is to sequentially perform simultaneous detection and tracking of objects described by the same set of keypoints P, in a video sequence $Z_k = \{z_1, \dots, z_k\}$, where z_k denotes the RGB image (an array of dimension $W \times H \times 3$) at discrete time instant k. The state space approach requires to specify a motion model, i.e., the evolution of the state $p(x_k|x_{k1})$, and a measurement model, i.e., the link between state and current measurement $p(z_k|x_k)$. The next two subsections describe the model of the object motion and the measurement likelihood function.

3.1.1 Motion Model

In this problem, since we are concerned with tracking the location of the tracked object, we select the state vector to be the position and velocity of the centre of the object in world frame. $x_k = [x_k, \dot{x_k}]^T$ where, $x_k = \{c_x, c_y, c_z\}$. Since the motion model of the object being tracked could possibly never follow a specific defined dynamics, for simplicity, we assume a constant velocity model given by,

$$x_k = F x_{k-1} + v_{k-1}$$

where F is a 6×6 transition matrix and v_{k-1} is the process noise, assumed to be white, zero-mean, Gaussian, with a covariance matrix Q. The predict step of the Particle Filter applies this motion model to all the particles, each of which is a hypothesis for the possible state of the tracked object. The method proposed could be easily extended to other motion models like constant acceleration model based on application.

3.1.2 Measurement Model

We leverage the visual sensory data along with the depth values from the RGB-D camera and integrate it in the measurement update step of the Particle Filter. Specifically, we do not use the entire image z_k as a measurement but rather we utilize the set of keypoints P of the object from the image computed inside the region bounding the object. This bounding box could be obtained using any of the real-time detection frameworks such as MobileNet-SSD, CSRT, etc.

In the first frame denoted by $k = 0$, a 3D model of the object in terms of its keypoints is created using the RGB-D data which is then stored as the reference ground truth to evaluate each particle hypothesis at future time-frame k. For each particle, a correlation value is computed which reflects the closeness of that particle's hypothesis with respect to the ground truth. Based on these correlation values, the particles are re-weighted and the resulting particle with highest weight is taken as the estimated pose of the object.

Creating Object Model: We assume here that the object tracked would be within the depth range of the RGB-D camera at the beginning of tracking. Given the image z_0 corresponding to the first frame and the first bounding box encompassing the object, a keypoint detector framework (such as SIFT or SURF) is used to get the 2D keypoints corresponding to the object inside the bounding box. Once we have the pixel coordinates of the keypoints $(u_i, v_i)_{i=1}^{N}$, where N is the number of detected keypoints, we get the 3D camera coordinates for each keypoint from the point-cloud data. We then get the relative positions of these 3D keypoints with respect to the camera coordinates centre of the bounding box $(c_x, c_y, c_z)_{cam}$ and store them in the set P. Since, the relative 3D positions of the keypoints from the center of the object should largely remain the same during tracking, the values in P should remain the same, making it an appropriate measure for the update step.

Computing Correlation: At any time instant k, when a new image is received from the RGB-D camera, we incorporate that as a measurement to update the weights of the particle in the update step of the filter. Based on the location of the tracked object w.r.t the camera, there arises three possible scenarios giving rise to different ways of updating the particle weights:

(A) Object in frame and within the depth range of camera: For this case, since the object is within the depth range, the actual measured location of the object in camera frame x_k^{cam}, called Object-origin for simplicity, can be directly obtained by taking the coordinates corresponding to the centre of the bounding box (c_u, c_v) from the point cloud.

$$x_{k,cam} = \{c_x, c_y, c_z\}_{cam}$$

Next, for each particle $\{w_k^i\}_{i=1}^{N_s}$ we use it's global estimate of the object location $x_k^{'}$ and convert it into the camera frame $x_{k,cam}^{'}$ using the camera's location and orientation w.r.t global frame. The L2 norm error between $x_{k,cam}^{'}$ and $x_{k,cam}$ reflects the deviation of particle estimate with respect to the measured value. Hence, the correlation in this case is just the inverse of this error.

$$Correlation3D = \frac{1}{||x_{k,cam}^{'} - x_{k,cam}||_2}$$

(B) Object in frame and out of depth range of camera: In this scenario, as we do not have the point-cloud data, we use the features from the RGB image z_k to compute correlation. We first detect the keypoints in z_k within the bounding box region provided by the detector. Feature matching with the object model stored in first frame provides the matching keypoints, K_{match}, between the two frames. For each particle $\{w_k^i\}_{i=1}^{N_s}$, we add the global pose estimate represented by that particle $x_k^{'}$ to each value in the set P, obtained from 3.1.2, corresponding to matched features K_{match}. The resulting values gives us the global coordinates of the object keypoints for the current time-frame k, assuming that the current particle represents the actual pose of the object. Using the camera's location and orientation w.r.t global frame and the camera intrinsics, we project each keypoint into the image space to get $K^{'}$. The case when the particle estimate is closer to the ground truth, the projection of these keypoints in the image space $K^{'}$ should be close to the actual detected and matched keypoints K_{match}. To evaluate this closeness, we again take the L2 norm error between K_{match} and $K^{'}$ and the inverse of that gives the correlation for this case. If the set K_{match} consists of m keypoints then,

$$Correlation2D = \frac{1}{\sum_{i=1}^{m} ||K_i^{'} - K_{i,match}||_2}$$

(C) Object out of camera frame: Here, since there is no measurement available to integrate into prediction, we only do the predict step which involves propagating the particle as per the motion model.

Alg. 1 gives a high-level pseudo-code of the Particle Filter implementation.

Algorithm 1 updateParticles

```
Require: $z_t, pointCloud, boundingBoxes, dt, particles \neq \phi$
  $propagateParticles(particles,\ dt)$
  $objectBox \leftarrow getBestBox(boundingBoxes)$
  if $objectBox$ is valid then
    $c_x, c_y \leftarrow getBoundingBoxCenter(objectBox)$
    $objectOrigin \leftarrow pointCloud(c_x, c_y)$
    $keypoints, descriptors \leftarrow getKeypoints2D(z_t, objectBox)$
    $keypointMatches \leftarrow matchKeypoints(objectModel,\ descriptors)$
    for all particles do
      if $origin$ is valid then
        $w_i \leftarrow getCorrelation3D(particles_i, objectOrigin)$
      else
        $w_i \leftarrow getCorrelation2D(particles_i, keypointMatches)$
      end if
    end for
  end if
  return $argmax_i(w)$
```

4 Results

4.1 Testing Methodology

We evaluated our approach using the "Princeton Tracking Benchmark" [8]. We found that most free RGB-D datasets were for testing segmentation and object reconstruction algorithms. For evaluating our approach we needed the ground truth position of the camera and object of interest. The Princeton dataset provides the bounding box annotations for some of its RGB-D videos, but not the ground truth of the object's position with respect to the camera. Hence for evaluating accuracy, we measured the error from the particle filter's highest scoring particle and the camera coordinates of the center of the bounding box. In case the bounding box is not detected for a frame, the ground truth is not evaluated as the object's position cannot be measured, as seen in the breaks in the lines for the Ground Truth in Figure 1. We used the `bear_front` dataset from [8] for evaluation. The camera is static in this dataset. We are using 100 particles for our tests and a diagonal matrix for covariance with values $[10^{-4}, 10^{-4}, 10^{-4}, 10^{-5}, 10^{-5}, 10^{-5}]$. Particles are resampled when the ratio of effective number of particles is less than 0.6.

The following tests were run on a laptop with an Intel i5-6300U CPU with 15.5GiB of DDR4 RAM.

4.2 Plots

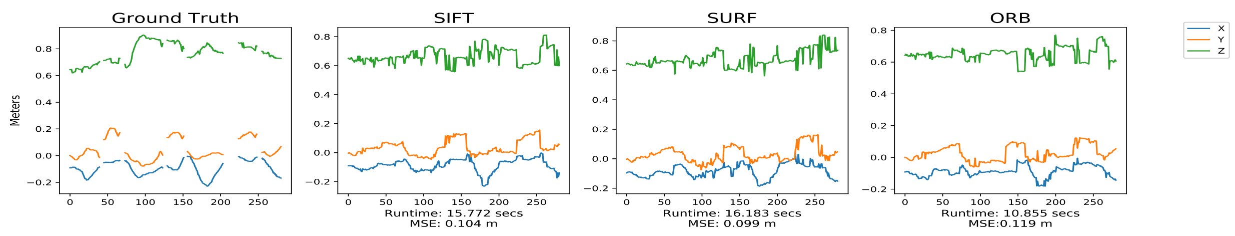

Figure 1: Performance of Camera Co-ordinate Estimation of Different Feature Descriptors

We implemented our approach in Python using the OpenCV libraries for keypoint matching. In Figure 1, we can see the performance of our particle filter when using different types of feature descriptors. In this figure, the filter only used depth information from the *first* frame to build the 3D model of the object. After this, the camera coordinates of the object is estimated by using the highest scoring particle of 100 particles, weighted using 2D correlation algorithm explained in Section 3.1.2. This is done to simulate the effect of the object moving out of the RGB-D camera's depth range and to put our `correlation2D` subroutine to

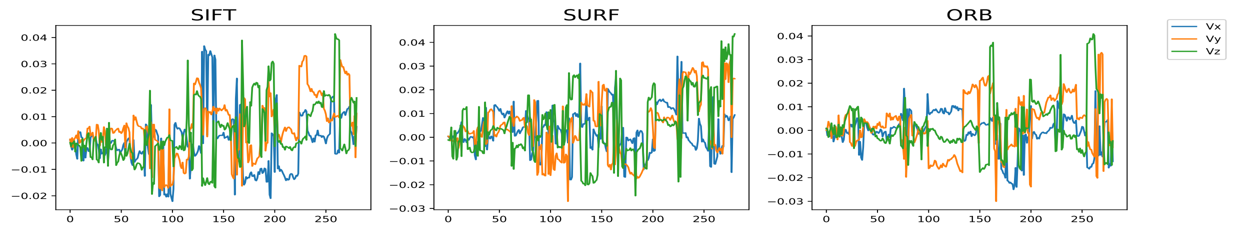

Figure 2: Predicted Velocities

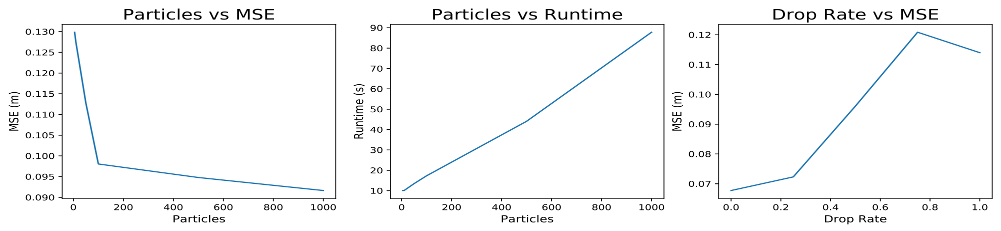

Figure 3: Effect of number of particles and drop rate of depth frames on performance

the test.

On the 281 frame `bear_front` dataset, the performance of using SURF and SIFT features was nearly identical. ORB features are 32% faster than SURF and SIFT, however it comes at the cost of having a 20% higher MSE from the ground truth position of the object. ORB features detected less keypoints than the other approaches. Our implementation is able to process frames at a rate of nearly 20fps for all feature descriptors. It is likely that a more optimized implementation written in C++, exploiting the 'embarrassingly parallel' nature of the particle filter will be able to process frames much quicker. The predicted velocities of the particles can be seen in 2. Since the object in the frame does not move consistently, the predicted velocites are quite noisy. We can also see that the particle filter is able to predict the position of the object when no bounding box is detected.

In Figure 3, we can see the impact of increasing the number of particles when using depth information only from the first frame of the dataset. The runtime of the algorithm increases linearly with increase in the number of particles used by the particle filter. However, increasing the number of particles used by the filter has diminishing returns as the MSE does not decrease significantly after using 100 particles.

We can also see the impact of the dropping depth frames in Figure 3. In the given dataset, depth information is available for each RGB frame, so we can simulate the object moving out of the depth camera's range by randomly choosing the frames for which depth information is available. The Drop Rate is the probability for which depth information the depth information is available. A Drop Rate of 0 means that the depth information for all the frames is used for position estimation, and a Drop Rate of 1 means that all the depth frames have been dropped. The first depth frame is always used for building the object model. We can see that our particle filter is able to make use of the extra depth frame detections for more accurate predictions.

5 Future Work and Conclusion

In terms of future work, we would like to experiment by using our algorithm on a real depth camera such as the Intel Realsense D435i. As this camera has an inbuilt IMU, we can estimate the camera's pose with respect to its initial position. This work can be extended to achieve accurate active tracking using a mobile robot such as a drone or a ground-based robot. The particle filter approach will also benefit from having an object model of different orientations of the object, so that it can detect the object from multiple angles. The particle filter's state can also be modified to use the constant acceleration model.

In summary, we have presented a method to estimate the position of an object with respect to an RGB-D camera by using a particle filter. We have discussed the limitations of using a depth camera and how using a correlation computed in pixel space can be used for computing the correlation of a particle. As this algorithm can run in realtime, it is suitable for mobile robots, such as drones.

References

[1] Z. Cao et al. “OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019).

[2] Vincent Lepetit and Francesc Moreno-noguer Pascal. “EP n P : An Accurate O (n) Solution to the P n P Problem”. In: April 2008 (2009), pp. 155–166. DOI: 10.1007/s11263-008-0152-6.

[3] Roman Bartak and Adam Vykovsky. “Any object tracking and following by a flying drone”. In: *Proceedings - 14th Mexican International Conference on Artificial Intelligence: Advances in Artificial Intelligence, MICAI 2015* (2016), pp. 35–41. DOI: 10.1109/MICAI.2015.12.

[4] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. “Tracking-learning-detection”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.7 (2012), pp. 1409–1422. ISSN: 01628828. DOI: 10.1109/TPAMI.2011.239. URL: http://vision.stanford.edu/teaching/cs231b_spring1415/papers/Kalal-PAMI.pdf.

[5] Runzhi Wang et al. “A New RGB-D SLAM Method with Moving Object Detection for Dynamic Indoor Scenes”. In: (2019).

[6] Alex Bewley et al. “Simple online and realtime tracking”. In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003.

[7] D. W. Eggert, A. Lorusso, and R. B. Fisher. “Estimating 3-D rigid body transformations: A comparison of four major algorithms”. In: *Machine Vision and Applications* 9.5-6 (1997), pp. 272–290. ISSN: 09328092. DOI: 10.1007/s001380050048.

[8] Shuran Song. “Tracking Revisited using RGBD Camera”. In: *Public Health* 9.C (1896), pp. 406–407. ISSN: 00333506. DOI: 10.1016/S0033-3506(96)80424-0. URL: https://vision.princeton.edu/projects/2013/tracking/paper.pdf.