Deep Quaternion Pose Proposals for 6D Object Pose Tracking

Mateusz Majcher Bogdan Kwolek✉
AGH University of Science and Technology, 30 Mickiewicza, 30-059 Kraków, Poland
{majcher,bkw}@agh.edu.pl

Abstract

In this work we study quaternion pose distributions for tracking in RGB image sequences the 6D pose of an object selected from a set of objects, for which common models were trained in advance. We propose an unit quaternion representation of the rotational state space for a particle filter, which is then integrated with the particle swarm optimization to shift samples toward local maximas. Owing to k-means++ we better maintain multimodal probability distributions. We train convolutional neural networks to estimate the 2D positions of fiducial points and then to determine PnP-based object pose hypothesis. A CNN is utilized to estimate the positions of fiducial points in order to calculate PnP-based object pose hypothesis. A common Siamese neural network for all objects, which is trained on keypoints from current and previous frame is employed to guide the particles towards predicted pose of the object. Such a keypoint based pose hypothesis is injected into the probability distribution that is recursively updated in a Bayesian framework. The 6D object pose tracker is evaluated on Nvidia Jetson AGX Xavier both on synthetic and real sequences of images acquired from a calibrated RGB camera.

1. Introduction

Pose estimation of known objects is essential task for robotic grasping and manipulation. It is one of the oldest problems in computer vision [16]. Early approaches to 6D object pose estimation aimed at finding correspondences between images and model instances [12]. More recent methods focus on generalizing to all instances within an object class by employing class-specific 3D keypoints [31]. Motivated by the success of large-scale image classification, last approaches are based on deep neural networks. In a seminal work [13] the object pose estimation is treated as a classification problem, where a neural network is trained to classify the image features into a discretized pose space. PoseCNN [36] was a first convolutional neural network (CNN) for direct regression of 6D object poses. Overall, in recent years there have been two main CNN-based approaches to object pose estimation, i.e. either regressing the 6D object pose from the image directly [36] or predicting 2D key-point locations in the image [23], from which the object pose can be determined by the PnP algorithm. In a two stage approach [23], in the first stage the centers of objects of interest are determined and afterwards deep neural networks are employed in order to determine the rotation of the object. Although recent methods that fit into the aforementioned research directions have turned out to be very successful at synthetic-to-real generalization [27], they still have not enough capabilities in terms of generalizing to novel unseen classes. Thus, considerable research efforts are devoted to training neural networks on large-scale datasets covering all kinds of objects, similar to ImageNet in the image classification domain [26].

Most existing methods for object pose estimation output a single guess of each object's pose [13, 36, 21, 26]. However, in context of robotic applications such approaches can be less useful as robots should be aware of pose uncertainty before taking an action. There are many environmental factors, such as illumination conditions, occlusions, object symmetry, not-enough texture as well as other factors such as insufficient training data that might lead to lower uncertainty of pose estimates. In case of lower uncertainty the robot should take information from the previous frame or eventually gather more information in order to reduce the uncertainty. Thus, methods for estimating the object pose, which are designed with robotics in mind should output a distribution of object poses rather than just a single estimate of the object pose. So far little work has been done in this area. In [24] a probabilistic modeling of the pose space for sequential state estimation has been studied. More recently, in [20] conditional probability distribution over orientations is used to update the hypothesis about actual object orientation. Modeling uncertainties in the form of continuous distributions over 3D object coordinates or bounding box coordinates have been proposed in [4] and [30], respectively.

Several robotic tasks such as visual object manipulation require online tracking. Although deep learning-based methods for object pose estimation can be executed relatively fast on modern GPUs, they re-estimate pose from

scratch for every frame, which is inherently redundant. Moreover, this results in less or even incoherent estimations of object poses over consecutive frames. Such data-driven techniques usually require real-world pose-annotated data. To reduce costs related to data annotations several recent methods focuses on training deep models on synthetic data, rendered on the basis of 3D models [13, 30]. Although techniques like domain adaptation and randomization can improve realism of synthetic data, training neural networks for object pose estimation on synthetic data can lead to much worse results in comparison to results achieved on relevant real data [13, 23]. It is also worth noting that photorealistic image synthesis requires accurately textured models, which in turn necessitates sophisticated engineering techniques to obtain desirable effects. Moreover, it is still common to train individual models for every newly encountered object instance. As lately shown, approaches relying on models that were trained for pose estimation of many objects resulted in deteriorated performance [37].

In this work we focus on embedded applications that can achieve fast adoption of a visual system for a novel set of objects to be tracked on RGB image sequences. We study quaternion pose distributions for tracking in image sequences the 6D pose of an object selected from a set of objects, for which common models were trained in advance. We propose an unit quaternion representation of the rotational state space for a particle filter, which is then integrated with the particle swarm optimization to shift samples toward local maximas. Thanks to k-means++ clustering we improve maintaining multimodal probability distributions. We train convolutional neural networks to estimate the 2D positions of fiducial points and then to determine PnP-based object pose hypothesis. A common Siamese neural network for all objects, which is trained on keypoints from current and previous frame is employed to guide the particles towards predicted pose of the object. Such a keypoint based pose hypothesis is injected into the probability distribution that is recursively updated in a Bayesian framework. The object is delineated by a common U-Net for all objects.

2. Quaternion Particle Filter. Quaternion Particle Swarm Optimization

Particle filters (PFs) allow robust estimation of hidden features of dynamical systems [15]. Due to their capability to deal with severe nonlinearities and non-Gaussian noise they are widely used in robotics and automotive industry [2].

Particle swarm optimization (PSO) [33] is a stochastic population-based optimization algorithm, which iteratively tries to ameliorate a candidate solution with respect to objective function. Substantial research efforts have been devoted to advance this global optimization algorithm. A selectively-informed approach [8] permits particles to choose different learning strategies based on their connections. In [5] the resampling from a particle filter has been used to improve the optimization performance of the PSO.

2.1. Quaternions

Quaternions are particularly appropriate within those areas of science where it is necessary to compose rotations with minimal computations. They permit reducing the number of parameters and operations in comparison to vector algebra. In [25], a fast quaternion-based PSO for pose estimation on RGB-D images has been proposed.

Mathematically, quaternions are members of a noncommutative division algebra. They can be regarded as numbers with one real part and three distinct imaginary parts: $\mathbf{q} = q_w + q_x\mathrm{i} + q_y\mathrm{j} + q_z\mathrm{k}$, where q_w, q_x, q_y, and q_z are real numbers, and $\mathrm{i}, \mathrm{j}, \mathrm{k}$ satisfy $\mathrm{i}^2 = \mathrm{j}^2 = \mathrm{k}^2 = \mathrm{ijk} = -1$, and $\mathrm{ij} = -\mathrm{ji} = \mathrm{k}$, $\mathrm{jk} = -\mathrm{kj} = \mathrm{i}$, $\mathrm{ki} = -\mathrm{ik} = \mathrm{j}$. The quaternion $\mathbf{q} = q_w + q_x\mathrm{i} + q_y\mathrm{j} + q_z\mathrm{k}$ can also be viewed as $\mathbf{q} = w + \mathbf{v}$, where $\mathbf{v} = q_x\mathrm{i} + q_y\mathrm{j} + q_z\mathrm{k}$. If we consider $\mathbf{v}$ as the 3D vector, then quaternion multiplication can be expressed using vector dot product and cross product of vectors. Every quaternion with unit magnitude that enforces the number of DoF to three, represents a rotation of angle θ about an arbitrary axis. If the axis passes through the origin of the coordinate system and has a direction given by the vector $\mathbf{n}$ with $|\mathbf{n}| = 1$, we can parameterize this rotation in the following manner:

$$\mathbf{q} = [q_w \;\; q_x \;\; q_y \;\; q_z] = \left[\cos(\tfrac{1}{2}\theta) \;\; \hat{\mathbf{n}}\sin(\tfrac{1}{2}\theta)\right] = [w \;\; \mathbf{v}] \quad (1)$$

The set of unit-length quaternions is a sub-group whose underlying set is named S^3. This set of unit quaternions corresponds to the unit sphere $\mathbb{S}^3$ in $\mathbb{R}^4$. As the quaternions $\mathbf{q}$ and $-\mathbf{q}$ represent identical rotation, only one hemisphere of $\mathbb{S}^3$ needs to be taken into account, and thus we choose the northern hemisphere $\mathbb{S}^3_+$ with $\mathbf{q} \geq 0$, which in turn is equivalent to $\theta \in [0, \pi]$.

The quaternion multiplication can be expressed as:

$$\begin{aligned}\mathbf{q}_0 \star \mathbf{q}_1 &= [w_0 \;\; \mathbf{v}_0][w_1 \;\; \mathbf{v}_1] \\ &= [w_0 w_1 - \mathbf{v}_0 \cdot \mathbf{v}_1 \;\; w_0\mathbf{v}_1 + w_1\mathbf{v}_0 + \mathbf{v}_0 \times \mathbf{v}_1]\end{aligned} \quad (2)$$

where $\times$ stands for vector cross product, $\cdot$ is vector dot product and $\star$ denotes quaternion multiplication. The logarithm of $\mathbf{q}$ is defined as follows:

$$logMap(\mathbf{q}) = logMap([cos(\alpha) \;\; \mathbf{n}sin(\alpha)]) \equiv [0 \;\; \alpha\mathbf{n}] \quad (3)$$

where $\alpha = \frac{1}{2}\theta$. It is worth noting that the $logMap(\mathbf{q})$ is not a unit quaternion. The exponential function is defined as:

$$expMap(\mathbf{p}) = expMap([0 \;\; \alpha\mathbf{n}]) \equiv [cos(\alpha) \;\; \mathbf{n}sin(\alpha)] \quad (4)$$

where $\mathbf{p} = [0 \;\; \alpha\mathbf{n}] = [0 \;\; (\alpha x \;\; \alpha y \;\; \alpha z)]$ with $\mathbf{n}$ as unit vector ($\|\mathbf{n}\| = 1$). By definition $expMap(\mathbf{p})$ always returns a unit quaternion. Exponential map has an advantage that it linearizes quaternions [9].

2.2. Quaternion Particle Filter

A particle filter (PF) is a Monte Carlo method that allows to solve such inverse problems. Recently, a dual quaternion filter for recursive estimation of rigid body motions has been proposed [18]. In this work we propose the unit quaternion representation of the rotational state space for a particle filter, which is then integrated with the particle swarm optimization responsible for moving the particles near the local maxims.

The state vector describing the 6D object pose comprises two parts: a quaternion as a description of rotations and translation vector in Euclidean space, which origin is in the camera coordinate system. Let us denote by $\mathbf{q}$ the unitary quaternion representing the rotation in time t, and by $\mathbf{z}$ the 3D translation in time t. The state vector assumes the following form: $\mathbf{x} = [\mathbf{q}\ \mathbf{z}]$, where $\mathbf{q}$ is a unitary quaternion and $\mathbf{z}$ is a 3D translation vector. To introduce the process noise in the quaternion motion of particle i, a three dimensional normal distribution with zero mean and covariance matrix C_r in the tangential space is applied as follows:

$$\mathbf{q}_i(t+1) = expMap(\mathcal{N}([0,0,0]^T, C_r)) \star \mathbf{q}_i(t) \tag{5}$$

where $\mathbf{q}_i(t)$ - orientation of particle i at time t, C_r - covariance matrix for rotation with standard deviations $(\gamma_{r1}\ \gamma_{r2}\ \gamma_{r3})$ on the diagonal, $\star$ - quaternion product (2) and $expMap$ - exponential function (4). The probabilistic motion model for the translation is as follows:

$$\mathbf{z}_i(t+1) = \mathbf{z}_i(t) + \mathcal{N}([0,0,0]^T, C_t) \tag{6}$$

Each particle i is represented as $s_i(t) = (\mathbf{x}_i(t), w_i(t))$, where $w_i(t)$ is the particle's weight. The weights are calculated on the basis of a probabilistic observation model and then used in the resampling of the particles. With the resampling the particles with large weights are replicated and the ones with negligible weights are eliminated.

2.3. Quaternion Particle Swarm Optimization

Particle Swarm Optimization (PSO) [14] is a global, derivative–free, population–based optimization method. It optimizes a problem by iteratively trying to improve a candidate solution with respect to a fitness measure. The optimal solution is sought by a population of particles exploring candidate solutions. During exploring the search space each particle is stochastically accelerated towards its previous best position (personal best) and towards the best solution of the group (global best). Every individual moves with its own velocity in the multidimensional search space, calculates its own best position and determines its fitness upon a fitness function $f(\mathbf{x})$. The objective function is employed to determine the best particles' locations as well as the global best location.

In an ordinary PSO every particle is initialized with a random position and velocity [14]. During exploration of the search space, every particle i updates its position that is affected by the best position $\mathbf{p}_i = [\mathbf{q}_{i,best}\ \mathbf{z}_{i,best}]$ determined so far and the global best position $\hat{\mathbf{g}} = [\mathbf{q}_{gbest}\ \mathbf{z}_{gbest}]$ found by the entire swarm.

The 3D position $\mathbf{z}$ and 3D velocity of the particle i in iteration k are determined as follows:

$$\begin{aligned}\mathbf{v}_i(k+1) = w\mathbf{v}_i(k) &+ c_1 r_1(\mathbf{z}_{i,best}(k) - \mathbf{z}_i(k)) \\ &+ c_2 r_2(\mathbf{z}_{gbest}(k) - \mathbf{z}_i(k))\end{aligned} \tag{7}$$

$$\mathbf{z}_i(k+1) = \mathbf{z}_i(k) + \mathbf{v}_i(k+1) \tag{8}$$

where $\mathbf{z}_i(k)$ is 3D position in iteration k, $\mathbf{v}_i(k)$ is velocity in iteration k, $\mathbf{z}_{i,best}(k)$ is the best 3D position found so far by the particle i, $\mathbf{z}_{gbest}(k)$ is the best 3D position of all particles, w is a positive inertia weight, c_1, c_2 are positive, cognitive and social constants, respectively, $r_{1,j}^{(i)}$ and $r_{2,j}^{(i)}$ are uniquely generated random numbers with the uniform distribution in the interval $[0.0,\ 1.0]$, generated in each iteration, for each dimension and independently for each particle. Spherical linear interpolation (SLERP) is used to obtain the object's angular velocity. The angular velocity update for the i-th particle is realized in the following manner:

$$\begin{aligned}\omega_i(k+1) =& w\omega_i(k) + c_1 r_1[2logMap(\mathbf{q}_{i,best}(k) \star \mathbf{q}_i^*(k))] \\ &+ c_2 r_2[2logMap(\mathbf{q}_{gbest}(k) \star \mathbf{q}_i^*(k))]\end{aligned} \tag{9}$$

where $\mathbf{q}_i(k)$ is rotation of particle i in iteration k, $\mathbf{q}_{i,best}(k)$ is the best rotation found so far by particle i, $\mathbf{q}_{gbest}(k)$ is the best rotation found so far by all particles and $logMap$ is logarithmic map (3). The rotation of the i-th particle is then updated in the following manner:

$$\begin{aligned}\mathbf{q}_i(k+1) = &\left[\cos\left(\frac{\|\omega_i(k+1)\|T_c}{2}\right),\right. \\ &\left.\sin\left(\frac{\|\omega_i(k+1)\|T_c}{2}\right)\frac{\omega_i(k+1)}{\|\omega_i(k+1)\|}\right]\mathbf{q}_i(k)\end{aligned} \tag{10}$$

where T_c is a parameter to scale the angular velocity. The number of iterations is set to three.

After determining $\mathbf{x}_i(k+1)$ using (8,10), $\mathbf{p}_i(k+1)$ is updated as follows:

$$\mathbf{p}_i(k+1) = \begin{cases} \mathbf{p}_i(k) & \text{if } f(\mathbf{x}_i(k+1)) \geq f(\mathbf{p}_i(k)) \\ \mathbf{x}_i(k+1) & \text{if } f(\mathbf{x}_i(k+1)) < f(\mathbf{p}_i(k)) \end{cases} \tag{11}$$

As topology with global best usually results in better performance on problems with small number of modes due to its faster convergence rate, such topology is used in this work.

3. Pose Tracking Using Quaternion PF-PSO

3.1. 6D Pose Tracking

Estimating the 6-DoF pose (3D rotations + 3D translations) of an object with respect to the camera is an impor-

tant problem due to potential applications in robotics. Automatic estimation of the object pose on RGB images is a difficult ill-posed problem. The discussed task has many important aspects that should be resolved to achieve robust object grasping, including object classification, object detection, object tracking, and finally, estimation of the 6D object pose. A lot of successful approaches have been developed in these directions [10, 6]. The pose of a calibrated camera can be estimated on the basis of a set of 3D points in the world and their corresponding 2D projections in the image by Perspective-n-Point (PnP) algorithm [7]. In such an approach, natural point features can be employed [17, 32]. In general, features can either encode image properties or can be learned. PoseCNN has been first convolutional neural network (CNN) for direct regression of 6DoF object poses [36]. In [13], the pose estimation is achieved by classification of image features in a discretized pose space. Generally, there are two main CNN-based approaches to 6D object pose estimation: regressing the 6D object pose from the image directly [36] or predicting 2D key-point locations in the image [23] and then using the PnP algorithm. In [21], a Pixel-wise Voting Network (PVNet) to regress pixel-wise unit vectors pointing to the keypoints and then using these vectors to vote for keypoint locations via RANSAC has been proposed. Although, several methods for pose estimation on single RGB images have been proposed, a number of successful approaches to 6D pose tracking is limited. There are a number of datasets for benchmarking the performance of algorithms for 6D object pose estimation, including OccludedLinemod [3], YCB-Video [36]. However, current datasets do not focus on 6D object tracking using RGB image sequences and they have been mainly designed for single-frame based pose estimation.

3.2. Algorithm for 6D Object Pose Tracking

On the basis of Q-PF and Q-PSO as the base ingredients we investigated an algorithm for 6-DOF object pose estimation and tracking on RGB images acquired from a calibrated camera. The object of interest is segmented in sequence of images using U-Net neural network. The segmented object is then fed to a neural network that estimates the 2D location of eight fiducial points on the object. Next, the 6D pose of the object is estimated by the PnP algorithm. It is then employed as a pose hypothesis during inference of the 6D pose. A Siamese neural network, which is trained on keypoints from current and previous frames is employed to predict the 6D object pose. The object pose prediction is employed as a pose hypothesis in the next frame. A quaternion particle filter (Q-PF) combined with a quaternion particle swarm optimization (Q-PSO) that were presented in Section 2 are employed to infer the posterior probability distribution of the object poses as well as the best 6D pose of the object. The observation model and objective function utilize the projected 3D model onto images in order to determine matching among the rendered object with the segmented object. The matching is calculated using object silhouette and distance transform-based edge scores. A hypothesis about 6D object pose that is determined on the basis of eight fiducial keypoints and the PnP algorithm is included in the probability distribution of the object poses. A second hypothesis is 6D object pose in the next frame. Assuming that this is multi-modal probability distribution a k-means++ algorithm is then executed to find dominant clusters in such a distribution. The quaternion particle swarm optimization is executed afterwards to seek modes in the probability distribution. It is also responsible for determining the best 6D object pose. The probability distribution that undergoes prediction to represent possible object poses in the next frame is represented by particles maintained by Q-PF, particles from two sub-swarms determined by the k-means++ as well as particles shifted towards high probability areas by the Q-PSO.

3.3. Siamese Neural Network for Pose Prediction

Figure 1 depicts architecture of neural network for prediction of the object pose. As it contains shared weights it is a Siamese-like neural network. The input are pairs of positions of eight keypoints in current frame and a previous one. The trained neural network predicts 3D positions and 3D rotations of the object in the next frame. The cost functions is calculated using 3D positions of the object in time $t-1$ and t, quaternions representing object rotations in time $t-1$ and t and delta translations from time $t-1$ to time t. The loss function is sum of the following components:

$$L = L_{T,t-1} + L_{Q,t-1} + L_{T,t} + L_{Q,t} + L_{DT1} + L_{DT2} \quad (12)$$

where $L_{T,t-1}$ and $L_{Q,t-1}$ denote the position and rotation loss components in time $t-1$. Same, $L_{T,t}$ and $L_{Q,t}$ represent position and rotation loss ingredient in time t. L_{DT1} and L_{DT2} stand for two delta translation loss components. First of them expresses discrepancy between ground truth translation with predicted translation. The second one calculates delta translation from predicted poses in time $t-1$, t and then calculates its discrepancy with the ground truth. The quaternions delivered by the network were normalized. Mean squared error was chosen to calculate all losses.

3.4. Quaternion PF-PSO with PnP-based Hypotheses and Siamese Neural Network-based 6D Pose Predictions

Given the particle set representing the posterior probability distribution in the previous frame, the particles are propagated according to a probabilistic motion model, the pose likelihoods on the basis of probabilistic observation model are calculated, and afterwards the particle weights are determined and a resampling is executed, as in ordinary particle

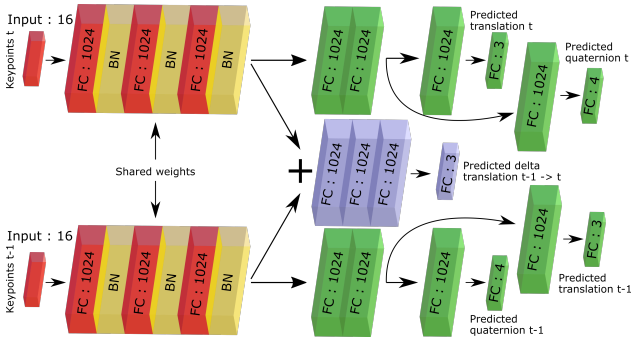

Figure 1: Architecture of Siamese neural network for prediction of 6D pose of the object.

filters. A particle with a small weight is replaced in such a resampled particle set by a particle with pose determined on the basis of keypoints and the PnP algorithm, see line #3 in below pseudo-code. After that, a particle with smallest weight is replaced by a particle with pose predicted by the Siamese neural network. Next, samples are clustered using k-means++ algorithm [1], which applies a sequentially random selection strategy according to a squared distance from the closest center already selected. Afterwards, a two-swarm PSO executes three iterations to find the modes in the probability distribution. Next, ten best particles are selected to form a sub-swarm, see lines #8-9 in pseudo-code. Twenty iterations are executed by such a sub-swarm to find better particle positions. The best global position returned by the discussed sub-swarm is used in visualization of the best pose. Finally, an estimate of the probability distribution is calculated by replacing the particle positions determined by the Q-PF with corresponding particle positions, which were selected to represent the modes in the probability distribution, see lines #6-7, and particles refined by the sub-swarm, see line #11. The initial probability distribution is updated by ten particles with better positions found by the Q-PSO algorithms and ten particles with better positions found by the sub-swarm, see line #12. The probabilistic observation model and objective function in the Q-PSO are based on matching among the rendered object with the segmented object. The matching is calculated using object silhouette and distance transform-based edge scores.

1 function select(n_best, X)
2 $\quad X_{sorted} \leftarrow$ quicksort(X) using $f(\mathbf{x})$
3 return $X_{sorted}[1...n\_best]$

Input: X_{t-1} - particle set (**x**=[**q z**])
1 $X_t \leftarrow$ propagate X_{t-1} using (5, 6)
2 $X_t \leftarrow$ PF(X_t) using $f(\mathbf{x})$
3 $\mathbf{x}_t^{PnP} \leftarrow PnP()$, replace worst $\mathbf{x} \in X_t$ with $\mathbf{x}_t^{PnP}$
4 $\mathbf{x}_t^{Siam} \leftarrow Siam()$, replace worst $\mathbf{x} \in X_t$ with $\mathbf{x}_t^{Siam}$
5 $X_t^{c1}, X_t^{c2} \leftarrow$ k-means++(X_t)
6 $\sim, X_t^{c1} \leftarrow$ QPSO($X_t^{c1}, 3$) using (8,10)
7 $\sim, X_t^{c2} \leftarrow$ QPSO($X_t^{c2}, 3$) using (8,10)
8 $X_t^{c1\_best} \leftarrow$ select($5, X_t^{c1}$)
9 $X_t^{c2\_best} \leftarrow$ select($5, X_t^{c2}$)
10 $X_t^{best} \leftarrow X_t^{c1\_best} \bigcup X_t^{c2\_best}$
11 $\mathbf{x}_{gbest}, X_t^{best} \leftarrow$ QPSO($X_t^{best}, 20$) using (8,10)
12 **foreach** $\mathbf{x}_1 \in X_t^{c1\_best} \bigcup X_t^{c2\_best} \bigcup X_t^{best}$ **do**
$\quad$ with $\mathbf{x}_1$ subst. corr. $\mathbf{x}_2 \in X_t$
13 return $\mathbf{x}_{gbest}, X_t$

The Q-PF-PSO presented above has been verified in various simulation experiments. Figure 2 depicts sample simulation results. In discussed experiment we simulated a scenario with rotating object about one of its axis. We assumed that an object rotates according to a probabilistic motion model. The object orientations are represented by red dots. The second image depicts the probability density, which was estimated using KDE after prediction of the samples in the Q-PF, whereas third one depicts the probability density after resampling. In the next images the densities in 1st, 2nd and 3rd iterations of Q-PSO are shown. In the discussed experiment, the measurements were contaminated by Gaussian random noise of zero value and a given covariance matrix. For visualization purposes relatively large standard deviations were utilized both in probabilistic motion and observation model.

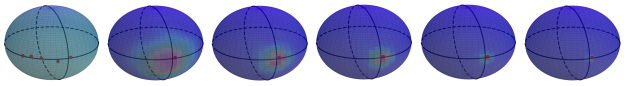

Figure 2: Simulation of Q-PF-PSO on toy data. From left to right: ground-truth marked as red dots, predicted distribution, distribution after resampling, distribution after 1-st, after 2-nd and after 3-rd iteration (best viewed in color).

4. Experimental Results

At the beginning we discuss computer simulations. Afterwards, we discuss evaluation metric for 6D pose estimation. Finally, we present experimental results.

4.1. Computer simulations

The proposed Q-PF, Q-PSO and Q-PF-PSO supported by a k-means++ are general algorithms and can be used in wide spectrum of applications. Figure 3 depicts sample plots that were obtained during tuning of Q-PF-PSO for tracking rotation of a real object, c.f. Section 4.3.

4.2. Evaluation Metric for 6D Pose Estimation

We evaluated the quality of 6-DoF object pose estimation using ADD score (Average Distance of Model Points)

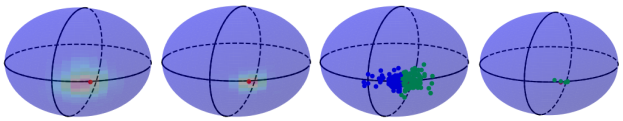

Figure 3: Simulation tests of Q-PF-PSO on real data. Prior distribution, distribution after resampling, clusters determined by k-means++, locations of particles after execution of PSO on two sub-swarms.

[11]. ADD is defined as average Euclidean distance between model vertices transformed using the estimated pose and the ground truth pose. This means that it expresses the average distance between the 3D points transformed using the estimated pose and those obtained with the ground-truth one. It is defined as follows:

$$ADD = \text{avg}_{x \in M} ||(Rx + t) - (\hat{R}x + \hat{t})||_2 \qquad (13)$$

where M is a set of 3D object model points, t and R are the translation and rotation of a ground truth transformation, respectively, whereas $\hat{t}$ and $\hat{R}$ correspond to those of the estimated transformation. This means that it expresses the average distance between the 3D points transformed using the estimated pose and those obtained with the ground-truth one. The pose is considered to be correct if average distance e is less than $k_e d$, where d is the diameter (i.e., the largest distance between vertices) of M and k_e is a predefined threshold (normally it is set to ten percent).

We determined also the rotation error on the basis of the following formula:

$$err_{\text{rot}} = \arccos((Tr(\hat{R}R^{-1}) - 1)/2) \qquad (14)$$

where Tr stands for matrix trace, $\hat{R}$ and R denote rotation matrixes corresponding to ground-truth and estimated poses, respectively.

4.3. Evaluation of Pose Tracking

All tracking scores presented below are averages of three independent runs of the algorithm with unlike initializations. At the beginning we evaluated our algorithm on freely available OPT benchmark dataset [35], which has been recorded for evaluation of algorithms for tracking 6D pose of the objects. In discussed benchmark dataset the image sequences have been recorded under various lighting conditions, different motion patterns and speeds using a programmable robotic arm. Table 1 presents the tracking scores that were achieved on the OPT dataset in FreeMotion scenario with the use as well as with no use of pose predictions by the Siamese neural network. As we can observe, considerable gains in the ADD scores can obtained thanks to Siemese neural network-based pose predictions. The discussed results were achieved using common Siamese neural network that has been trained for all objects.

Table 1: Tracking scores [%] achieved by our algorithm on OPT dataset [35] in tracking 6D pose of House and Ironman in FreeMotion scenario (nS - no Siamese).

view, ADD [%]	House (nS)	House	Ironman (nS)	Ironman
Behind, 10%	85±2.07	82±2.53	64±2.29	77±3.06
Behind, 20%	99±0.51	97±1.68	91±1.62	95±2.06
Left, 10%	81±1.52	76±2.58	35±5.56	42±3.37
Left, 20%	97±1.34	98±1.34	66±6.77	69±3.82
Right, 10%	55±5.13	75±2.02	46±5.15	56±4.34
Right, 20%	74±7.20	96±1.80	73±6.73	79±5.04
Front, 10%	53±3.66	82±1.14	55±6.02	68±3.45
Front, 20%	81±5.40	97±1.10	73±5.94	83±3.29
Average, 10%	68	79	50	61
Average, 20%	88	97	76	82

Afterwards, we compared results achieved by our algorithm with a neural network trained for all objects with results that were obtained by a neural network trained for each object. Table 2 presents the tracking scores achieved in 6D pose tracking of House, Ironman and Jet objects in FreeMotion scenario. Although the tracking scores achieved by the algorithm with a Siamese neural network trained for all object are slightly worse than these achieved with the use of a separate network for each object, the difference between tracking scores is not high.

Table 2: Tracking scores [%] achieved by our algorithm in tracking 6D pose of House, Ironman and Jet in FreeMotion scenario.

	Siamese for all			Siamese for each		
tracking score [%]	House	Iron.	Jet	House	Iron.	Jet
Behind, ADD 10%	82	77	74	85	73	80
Behind, ADD 20%	97	95	86	99	93	90
Left, ADD 10%	76	42	39	80	48	33
Left, ADD 20%	98	69	66	97	68	55
Right, ADD 10%	75	56	60	78	58	61
Right, ADD 20%	96	79	84	96	79	86
Front, ADD 10%	82	68	61	82	69	68
Front, ADD 20%	97	83	86	98	87	89
Average, ADD 10%	79	61	59	81	62	60
Average, ADD 20%	97	82	80	97	82	80

Table 3 contains AUC scores achieved by recent methods in 6D pose tracking of six objects in the FreeMotion scenario. As we can observe, our algorithm achieves better results in comparison to results achieved by PWP3D, UDP and ElasticFusion. In comparison to results attained by algorithm [29] our method outperforms it in many cases. However, the discussed method delivers a single guess of each object's pose, whereas our method delivers best poses together with probability distributions.

Afterwards, we conducted experiments on our dataset [19]. For evaluation of 6D pose tracking we selected the sequences #2 in which every object moves from left to right, simultaneously makes rotations from 0 to 180°, and after reaching the right side of the image, the object moves back

Table 3: AUC scores on OPT Dataset [35] in FreeMotion scenario, compared to results achieved by recent methods.

AUC score [%]	House	Ironman	Jet	Bike	Chest	Soda
PWP3D [22]	3.58	3.92	5.81	5.36	5.55	5.87
UDP [4]	5.97	5.25	2.34	6.10	6.79	8.49
ElasticFusion [34]	2.70	1.69	1.86	1.57	1.53	1.90
Reg. G-N. [29]	10.15	11.99	13.22	11.90	11.76	8.86
w/o Siamese	11.52	9.26	9.19	10.81	6.93	6.61
Siamese for each	**13.68**	10.59	10.37	**12.36**	7.80	8.60
Siamese for all	13.27	10.32	10.33	11.88	7.60	**8.90**

from right to left, simultaneously makes full rotation about its axis. The training of neural networks both for object segmentation/detection and 6D pose estimation was done on synthetic images only. Table 4 presents experimental results that were obtained by our algorithm.

Figure 4 depicts ADD scores over time that were obtained by the Q-PF-PSO algorithm. As we can observe, slightly bigger errors were achieved for 0° camera view for the extension and the multimeter. The larger errors are due to similar appearances of the objects from the side view. For 10% ADD the pose predictions by the Siamese neural network lead to considerable improvements of tracking scores. Although, the tracking scores achieved by the algorithm with Siamese neural network trained for all objects are smaller, the drop in the performance is relatively small. The advantages of using a single neural network for all objects are considerable. Moreover, this in turn opens several new research possibilities.

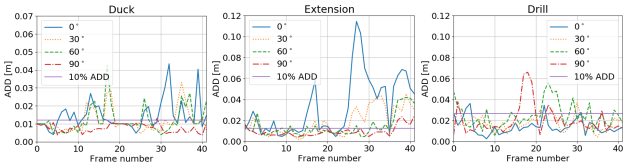

Figure 4: ADD scores over time for images from sequence #2, obtained by Q-PF-PSO with Siamese neural network predicting object pose (plots best viewed in color).

Figure 5 depicts rotation errors over time on real images, which have been obtained by the Q-PF-PSO with neural network predicting the object pose over time. As we can observe, the largest errors were obtained for the extension, which is symmetric and texture-less object. Somewhat larger errors were observed for 30° camera view. The experimental results presented above were achieved on the basis of eight fiducial points, which were determined by a neural network, trained separately for each object. This could indicate the limited potential of the proposed approach. However, it is worth noting that the proposed approach is universal as it can be used for points that are discovered in semi-supervised or unsupervised learning.

In order to segment all objects we trained a single U-Net

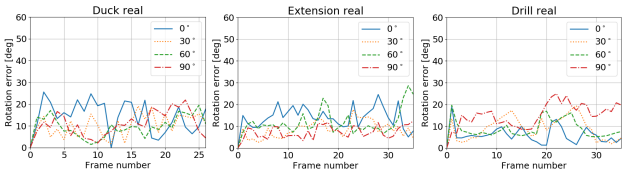

Figure 5: Rotation errors over time for real images, obtained by Q-PF-PSO using object poses that were predicted by the Siamese neural network.

on 900 images from the OPT benchmark. A single U-Net has been trained on 1800 images in order to segment all six objects from our dataset. The Dice scores were higher than 95% for all objects. The neural networks for estimation of positions of keypoints were trained on 300 images and evaluated on 50 images. We trained separate Siamese neural networks for OPT and our dataset. They were trained in 15 epochs, batch size set to 256 using Adam optimizer with l_r set to $1e{-}4$. It has been trained on $1e6$ pairs of positions of eight keypoints. The keypoint positions were generated on the basis of projections of 3D keypoints of the objects. The pairs were generated with the starting pose in the range [-180°, 180°] with 45° step on every axis. Translation was sampled from uniform distribution in interval [-30 cm, 30 cm] with step 15 cm for all axes. Z axis was fixedly increased to fit the object in the image. The second pose of pair was then generated by changing randomly the starting rotations on the basis of (5) and changing randomly the translation through sampling from uniform distribution [-5 cm, 5 cm] with 2 cm step for all axes. It is obvious that the use of trajectories, or even keypoints from three consecutive frames might lead to better results or similar result but using fever number of particles.

The complete system for 6D pose estimation has been evaluated on Nvidia Jetson AGX Xavier board. The Jetson AGX Xavier has 512-core Volta GPU with Tensor Cores. It offers more than 20x the performance and 10x the energy efficiency of the Nvidia Jetson TX2. Table 5 presents running times that were compared with times obtained on a PC equipped with AMD Ryzen 7 2700, GeForce RTX 2060. The evaluations were performed by setting the Jetson AGX Xavier board to maximum performance mode (MAXN0). We used Jetpack 4.3, CUDA 10 and Tensorflow 2.1. We have optimized the weights of neural networks with the TensorRT library (6.0) with default settings. As we can observe, thanks to optimized code the U-Net executes on the Jetson in about two times shorter time. The speed-up for the Siamese neural network is even larger. The time needed for determination of the keypoints on the Jetson is somewhat larger in comparison to time needed on the PC. The discussed operations are executed in 0.04 sec. on the Jetson, i.e. with about 25 Hz. Thanks to implementation of the k-means++ in CUDA and executing it on the GPU, al-

Table 4: ADD scores [%] achieved by our algorithm with and without Siamese.

tracking score [%]	0°, ADD 10%	0°, ADD 20%	30°, ADD 10%	30°, ADD 20%	60°, ADD 10%	60°, ADD 20%	90°, ADD 10%	90°, ADD 20%	Avg., ADD 10%	Avg, ADD 20%
drill w/o Siam.	88	98	88	98	68	93	77	98	80	97
drill with Siam. sep.	92	98	87	100	70	93	83	93	83	96
drill with Siam. com.	90	98	90	99	78	93	77	84	84	94
frog w/o Siam.	71	82	87	98	66	79	38	57	65	79
frog with Siam. sep.	81	87	75	81	83	93	66	82	76	86
frog with Siam. com.	81	87	81	89	76	87	63	88	75	88
pig w/o Siam.	45	74	52	73	63	83	85	93	61	81
pig with Siam. sep.	53	73	61	70	64	79	93	100	68	80
pig with Siam. com.	53	73	56	83	64	79	91	93	66	82
duck w/o Siam.	52	94	78	86	73	79	86	100	72	90
duck with Siam. sep.	55	88	79	88	75	84	93	100	75	90
duck with Siam. com.	53	79	84	91	77	90	93	100	77	90
ext. w/o Siam.	23	35	58	71	62	75	75	98	54	70
ext. with Siam. sep.	37	50	58	77	70	83	86	97	63	76
ext. with Siam. com.	40	49	58	68	70	83	81	97	62	74
mult. w/o Siam.	15	26	75	93	85	96	94	100	67	79
mult. with Siam. sep.	17	26	76	96	84	98	98	99	69	80
mult. with Siam. com.	15	28	78	98	84	98	93	100	68	81

most five times speed-up has been obtained in comparison to executing the k-means++ on NVIDIA Carmel ARMv8.2 CPU. In the current unoptimized GPU implementation of the PSO it is executed on the Jetson in almost two times larger time than on the PC. Summarizing, the Jetson-based implementation of the algorithm runs at about 5 Hz, i.e. at about the same speed as on the PC. The most computationally demanding operation is matching between the projected 3D model and image observations, supported by an edge distance transform. The functions mentioned above are implemented using CUDA parallel programming environment. They can be further speed up with more advanced parallel processing such as CUDA-OpenGL interoperability, which has been shown to render one thousand images in 100 milliseconds [28].

Table 5: Running times on Jetson AGX Xavier [sec.]

	PC	Jetson
U-Net	0.040	0.020
Keypoints	0.035	0.040
Siamese	0.030	0.007
k-means++	0.005	0.010
PSO 200p. 3 iter.	0.037	0.060
PSO 10p. 10 iter.	0.026	0.040
overheads	0.017	0.023
Total	0.190	0.200

5. CONCLUSIONS

In this paper, we showed that Siamese neural network can deliver pose predictions over time, which considerably improve the performance of the object tracking. We demonstrated experimentally that the tracking accuracy does not drop significantly if one Siamese neural net trained for several objects is used instead of the neural network trained for each object separately. We proposed an unit quaternion representation of the rotational state space for particle filter hybridized with the particle swarm optimization. The 6D object pose tracker was evaluated both on synthetic and real sequences of images acquired from a calibrated RGB camera. The evaluations were done on freely available OPT benchmark dataset as well as on our dataset that contains both real and synthesized images of six texture-less objects. We demonstrated experimentally that thanks to use of pose predictions by the Siamese neural network the tracking performance is much better. One of the advantages of our approach is that in contrast to recent approaches, our algorithm delivers probability distribution of object poses instead of a single object pose guess. On Nvidia Jetson AGX Xavier the neural networks for object segmentation, fiducial keypoints determination and pose prediction are executed with 25 Hz, whereas the whole algorithm is running at about 5 Hz. In ongoing work the keypoint extraction will be done in a semi-supervised manner.

Acknowledgement

This work was supported by Polish National Science Center (NCN) under a research grant 2017/27/B/ST6/01743.

References

[1] D. Arthur and S. Vassilvitskii. K-means++: The Advantages of Careful Seeding. In *Proc. ACM-SIAM Symp. on Discrete Algorithms*, pages 1027–1035, 2007. 5

[2] K. Berntorp and S. Di Cairano. Particle Filtering for automotive: A survey. In *22th Int. Conf. on Inf. Fusion*, pages 1–8, 2019. 2

[3] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother. Learning 6D object pose estimation using 3D object coordinates. In *ECCV*, pages 536–551, 2014. 4

[4] E. Brachmann, F. Michel, A. Krull, M. Yang, S. Gumhold, and C. Rother. Uncertainty-driven 6D pose estimation of objects and scenes from a single RGB image. In *CVPR*, pages 3364–3372, 2016. 1, 7

[5] Q. Cheng, X. Han, T. Zhao, and Sarma Y. Improved Particle Swarm Optimization and neighborhood field optimization by introducing the re-sampling step of Particle Filter. *J. of Industrial & Management Opt.*, 15:177–198, 2019. 2

[6] X. Deng, A. Mousavian, Y. Xiang, F. Xia, T. Bretl, and D. Fox. PoseRBPF: A Rao-Blackwellized Particle Filter for 6D Object Pose Tracking. In *Robotics: Science and Systems (RSS)*, 2019. 4

[7] M. Fischler and R. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Commun. ACM*, 24(6):381–395, 1981. 4

[8] Y. Gao, W. Du, and G. Yan. Selectively-informed particle swarm optimization. *Scientific Reports*, 5(1), 2015. 2

[9] F. Grassia. Practical parameterization of rotations using the exponential map. *J. of Graphics Tools*, 3(3):29–48, 1998. 2

[10] K. He, G. Gkioxari, P. Dollar, and R. Girshick. Mask R-CNN. In *ICCV*, pages 2980–2988, 2017. 4

[11] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab. Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes. In *ACCV 2012*, pages 548–562, 2013. 6

[12] D. Huttenlocher and S. Ullman. Recognizing solid objects by alignment with an image. *Int. J. of Computer Vision*, 5(2):195–212, 1990. 1

[13] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab. SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again. In *IEEE Int. Conf. on Computer Vision*, pages 1530–1538, 2017. 1, 2, 4

[14] J. Kennedy and R. Eberhart. Particle Swarm Optimization. In *Proc. of IEEE Int. Conf. on Neural Netw.*, pages 1942–1948, 1995. 3

[15] A. Kutschireiter, C. Surace, H. Sprekeler, and J.-P. Pfister. Nonlinear Bayesian filtering and learning: a neuronal dynamics for perception. *Scientific Reports*, 7(1), 2017. 2

[16] R. Lawrence. *Machine Perception of 3D Solids*. PhD thesis, Massachussets Inst. of Technology, MIT, 6 1965. 1

[17] V. Lepetit, J. Pilet, and P. Fua. Point matching as a classification problem for fast and robust object pose estimation. In *CVPR*, pages 244–250, 2004. 4

[18] K. Li, F. Pfaff, and U. D. Hanebeck. Unscented dual quaternion particle filter for SE(3) estimation. *IEEE Control Systems Letters*, 5(2):647–652, 2021. 3

[19] M. Majcher and B. Kwolek. 3D model-based 6D object pose tracking on RGB images using particle filtering and heuristic optimization. In *VISAPP*, pages 690–697, vol. 5, 2020. 6

[20] Z. C. Márton, S. Türker, Ch. Rink, M. Brucker, S. Kriegel, T. Bodenmüller, and S. Riedel. Improving object orientation estimates by considering multiple viewpoints. *Aut. Rob.*, 42(2):423–442, 2017. 1

[21] S. Peng, Y. Liu, Q. Huang, X. Zhou, and H. Bao. PVNet: Pixel-Wise Voting Network for 6DoF Pose Estimation. In *IEEE Conf. CVPR*, pages 4556–4565, 2019. 1, 4

[22] V. Prisacariu and I. Reid. PWP3D: Real-Time Segmentation and Tracking of 3D Objects. *Int. J. Comput. Vision*, 98(3):335–354, 2012. 7

[23] M. Rad and V. Lepetit. BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3D poses of challenging objects without using depth. In *IEEE ICCV*, pages 3848–3856, 2017. 1, 2, 4

[24] S. Riedel, Z.-C. Marton, and S. Kriegel. Multi-view orientation estimation using Bingham mixture models. In *IEEE Int. Conf. on Aut., Quality and Testing, Robotics*. IEEE, 2016. 1

[25] S. Rosa, G. Toscana, and B. Bona. Q-PSO: Fast quaternion-based pose estimation from RGB-D images. *J. of Intelligent & Robotic Systems*, 92(3):465–487, 2018. 2

[26] C. Sahin, G. Garcia-Hernando, J. Sock, and T.-K. Kim. A review on object pose recovery: From 3D bounding box detectors to full 6D pose estimators. *Image and Vision Comp.*, 96:103898, 2020. 1

[27] H. Su, Ch. Qi, Y. Li, and L. Guibas. Render for CNN: Viewpoint estimation in images using CNNs trained with rendered 3D model views. In *IEEE ICCV*, 2015. 1

[28] Z. Sui, L. Xiang, O. Jenkins, and K. Desingh. Goal-directed robot manipulation through axiomatic scene estimation. *The Int. J. of Robotics Research*, 36(1):86–104, 2017. 8

[29] H. Tjaden, U. Schwanecke, E. Schmer, and D. Cremers. A region-based Gauss-Newton approach to real-time monocular multiple object tracking. *IEEE Trans. on PAMI*, 41(8):1797–1812, 2019. 6, 7

[30] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. In *Proc. 2nd Conf. on Robot Learn.*, volume 87, pages 306–316, 2018. 1, 2

[31] H.-Y. Tseng, S. De Mello, J. Tremblay, S. Liu, S. Birchfield, M.-H. Yang, and J. Kautz. Few-shot viewpoint estimation. In *BMVC*, page 39, 2019. 1

[32] J. Vidal, C. Lin, and R. Mart. 6D pose estimation using an improved method based on point pair features. In *Int. Conf. on Control, Aut. and Robotics*, pages 405–409, 2018. 4

[33] D. Wang, D. Tan, and L. Liu. Particle swarm optimization algorithm: An overview. *Soft Comp.*, (2):387–408, 2018. 2

[34] T. Whelan, R. Salas-Moreno, B. Glocker, A. Davison, and S. Leutenegger. ElasticFusion. *Int. J. Rob. Res.*, 35(14):1697–1716, 2016. 7

[35] P. Wu, Y. Lee, H. Tseng, H. Ho, M. Yang, and S. Chien. A benchmark dataset for 6DoF object pose tracking. In *Int. Symp. on Mixed & Aug. Reality*, pages 186–191, 2017. 6, 7

[36] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox. PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. In *IROS*, 2018. 1, 4

[37] S. Zakharov, I. Shugurov, and S. Ilic. DPOD: 6D pose object detector and refiner. In *ICCV*, pages 1941–1950, 2019. 2