Monocular Pose Estimation for Human-Robot Co-Localization

Christopher Tibaldo[1] tibaldoc@ethz.ch
Jonas Bohn[1] bohnj@ethz.ch
Mae Yamaguchi[1] myamaguchi@ethz.ch
Michel Zeller[1] mizeller@ethz.ch

[1]ETH Zurich

Abstract

In a human-robot interaction scenario, it is crucial for the different agents to be able to precisely localize each other in the shared environment, i.e. to visualize a planned trajectory. However, feature-based visual registration is an arduous task because feature matching between different camera streams is computationally expensive and challenging due to different points-of-view. In this paper, we propose employing a monocular pose-estimation algorithm to obtain an initial guess of the spatial relationship, that can serve as a good prior for subsequent multi-agent pose-graph optimisation.

1. Introduction

Human-robot interaction in shared work spaces has become increasingly prevalent across various domains, i.e. manufacturing, logistics, healthcare to name a few. To enable such interactive scenarios, it is vital to accurately determine the relative positions of robots and humans to ensure efficient and save cooperation.
Current localization methods primarily rely on identifying matching key-points between the different video streams. Visual features are captured using cameras mounted on both robots and humans, i.e. using Mixed Reality (MR) headsets such as the Microsoft HoloLens. Key-point matching algorithms, like Scale-Invariant Feature Transform (SIFT) [11] are commonly used to extract distinctive features. However, diverse camera positions and resulting variations in perspectives make finding matching features difficult, leading to time-consuming computations.
In this work, we present a novel approach to expedite the localization process by leveraging deep learning-based object pose estimation algorithms, to accurately determine the pose of a robot as seen from a human wearing a HoloLens. The used approach aims to overcome the limitations of traditional feature matching techniques by learning the unique visual characteristics of the robot from a high-quality Neural Radiance Field (NeRF) [13] model visualized in fig 1 and perform pose estimation afterwards.
To facilitate experimentation and evaluation, we introduce a modular synthetic data pipeline using BlenderProc2 [3]. Our dataset of around 2000 frames, following the BOP format ([6]), is designed for training pose estimation algorithms on Boston Dynamics Spot in various scenes. We also provide the data conversion scripts to integrate the dataset with a slightly modified version of OnePose++ [4] for training pose estimation models. Our contributions include a high-quality NeRF model of Spot, a versatile data pipeline, a dataset in BOP format, data conversion scripts, and a tailored OnePose++ algorithm for the synthetic data.

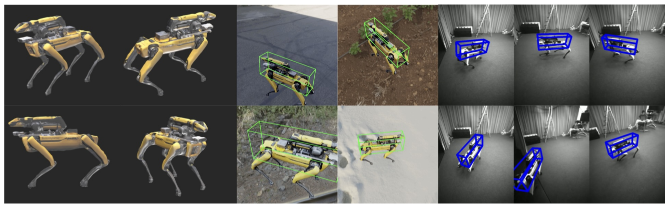


Figure 1. From NeRF model to real-world Pose Estimation

2. Related Work

6-DoF object pose estimation. The objective of 6 degrees-of-freedom (DoF) pose estimation is to determine the position and orientation of a rigid object within a three-dimensional (3D) scene. Recent advancements in deep learning have significantly improved the performance of object pose estimation algorithms. These algorithms have achieved impressive results on LineMOD [5], a widely-used benchmark dataset for 6-DoF pose estimation, which consists of RGB or RGB-D data. Due to the widespread availability and cost-effectiveness of RGB data, this work specifically concentrates on methods that utilize single RGB images as input.

CAD-model based methods. Object pose estimation methods often require CAD models for training or rendering purposes. These methods can be classified based on their ability to generalize. Instance-level methods employ

convolutional neural networks (CNNs) to directly map observed RGB images to object poses [7, 9, 12, 14, 18], they utilize 2D keypoint detection and compute poses using 2D-3D correspondences with the Perspective-n-Point (PnP) algorithm. Category-level methods [1, 2, 17], on the other hand, overcome the need for CAD models from the same category during testing by learning a shape prior specific to the representative object of that category. However, these category-level methods face limitations in their generalizability to unseen categories or objects with different appearances or shapes than the prior object within the category. While the utilization of CAD models in object pose estimation yields promising results for trained instances or categories, the unavailability of CAD models in certain scenarios has driven the emergence of more dynamic methods that are better suited for practical applications.
CAD-model free methods. 6-DoF object pose estimation without the usage of a CAD model gained relevance, as they can be applied on arbitrary objects without any previous knowledge. This makes such methods much more feasible to use in real-life scenarios. OnePose [16], OnePose++ [4] and Gen6D [10] only require a set of reference images with some ground truth poses to perform pose estimation on unseen objects. Gen6D takes on three main steps, detection of the object in the query image, selecting the reference image with the most similar viewpoint and refining the pose to perform accurate pose estimation. Nonetheless, the Gen6D relies on an accurate object detection and has some problems in estimating the pose if the object is occluded. OnePose and OnePose++ both create a point cloud of the object from the reference images. OnePose is only using repeatable image keypoints for structure-from-motion (SfM) to create a sparse point-cloud, while the novel OnePose++ proposes a key-point-free SfM method to reconstruct a semi-dense point cloud of the object. They then directly perform 2D-3D correspondence matching between the query image and the reconstructed point cloud. OnePose++ achieves similar performance on the LineMOD dataset as CAD-model-based methods, which emphasizes the eligibility for real-world applications. Another approach was proposed by NeRF-Pose [8], where instead of a point cloud they reconstruct the object using NeRFs and afterwards perform dense 2D-3D correspondence matching with PnP+RANSAC to estimate the pose of the object. However, the NeRF-Pose method is not able to generalize to unseen data. Our work relies mainly on OnePose++ and combines it with the idea from NeRF-Pose in using NeRFs for creating dense representations of the target object.

3. Method

In the following, we present a technique to determine the pose of a robot as perceived by a human. The whole pipeline can be seen in fig 2. Initially we imposed some natural constraints on our problem-statement, namely :

- The robot is ground-based and smaller than the human.
- The gaze direction is always towards the robot.
- The robot is close to the human.
- The agents share a structured indoor environment.

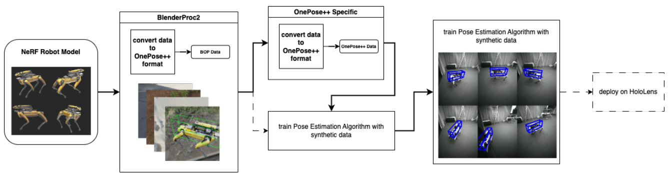

NeRF Robot Model
BlenderProc2
convert data to OnePose++ format
BOP Data
OnePose++ Specific
convert data to OnePose++ format
OnePose++ Data
train Pose Estimation Algorithm with synthetic data
train Pose Estimation Algorithm with synthetic data
deploy on HoloLens

Figure 2. Pipeline Structure

These assumptions allow to easily re-create an approximate real-world scenario using a procedural Blender pipeline for photo-realistic rendering.

3.1. Data

Deep learning research over time has demonstrated that learning good models require large volumes of training data to perform well on real-world applications such as augmented reality, autonomous driving or object pose estimation in our case. Training an object pose estimation network requires accurate pose labels for each frame. Sec. 3.1.1 covers the used method to extract pose labels from real-world data and the limitations faced. Sec. 3.1.2 on the other hand discusses synthetically created images and labels, which could potentially provide a lot of data in a cost-effective way with the risk of sacrificing realism for real-world application.

3.1.1 Real-World Data

Sensor readings from the HoloLens deadset and Spot were recorded using ROS (Robotic Operating system). ROS is a commonly used library to control robotic systems. The saved .bag files contain all messages sent by each of the agents sensors, including camera feeds.

Working with real-world data posed several challenges. The first challenge was co-localizing the recordings in a shared frame of reference. After setting up a Docker environment, capable of running ROS we passed the recorded data to Maplab [15].

This is a tool which can process recordings as well as real-time sensor data to reconstruct a 3D model of the physical environment, in which the robotic agents are moving. By matching shared SIFT features from the two video-streams we managed to establish the global position of the two. This comprises the first step of establishing a real-world data ground truth (see fig 3).

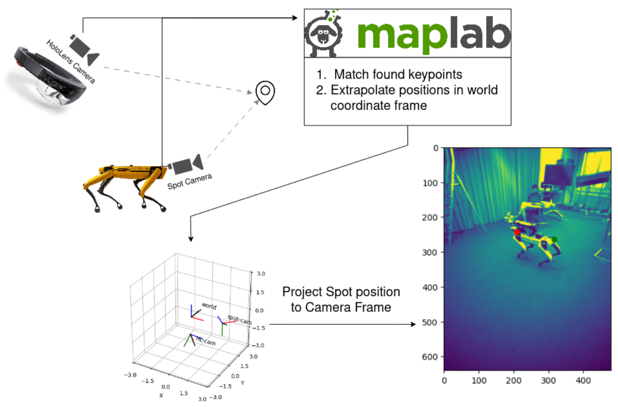

maplab
1. Match found keypoints
2. Extrapolate positions in world coordinate frame
HoloLens Camera
Spot Camera
Project Spot position to Camera Frame

Figure 3. From Camera captures, to global positions, to back-projection

Once the global positions of the two agents are known, it is necessary to back-project the robot's position into frame of reference of the human-worn camera.

```python
def get_projected_point(pq_spot,
        pq_HL, p=[0., 0., 0.]):
  T_G_spot=pytr.transform_from_pq(pq_spot)
  T_G_HL = pytr.transform_from_pq(pq_HL)
  T_HL_spot=invert_transform(HL_T_imu_cam)
    @ invert_transform(T_G_HL)
    @ (T_G_spot)
    @ (spot_T_imu_cam)
  p_in_HL_frame = T_HL_spot
    @ np.array([p[0],p[1],p[2],1.])[:,None]
  assert p_in_HL_frame[3] == 1.
  point2d = K @ Id @ p_in_HL_frame
```

Figure 4. Code used to project point from world coordinate frame to camera frame

The function in fig [4] takes the global position of Spot and HoloLens expressed as $\text{pq\_spot}, \text{pq\_HL} \in [\text{3D position}, \text{Quaternion}]$. These vectors are turned into transformation matrices, which are then multiplied with the transformation matrices representing the offset between the camera and the device's inertial measurement unit, the location of the origin. After having obtained the transformation matrix between the robot's coordinate frame and the HoloLens coordinate frame we project the origin of the robot coordinate frame ($[0, 0, 0]$). This point is then multiplied with the intrinsic parameters of the HoloLens (represented by K) to obtain the position projected into the image.

Unfortunately, there is an underlying bug somewhere in the code-base of [15]. After numerous hours of debugging we were not able to solve the reprojection step. Thus we did not have ground truth for the real-world data.

3.1.2 Synthetic Data

Robot Model The most crucial element of the synthetic data is the robot model, as it is the target object we want to predict the pose of. Initially we used a publicly available CAD model of Spot from the Spot ROS driver repository[1]. Some frames from the initial synthetic training data can be seen in fig [5].

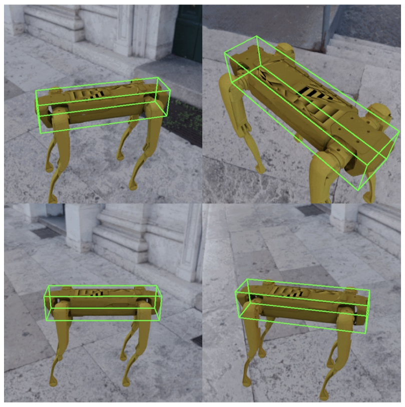


Figure 5. CAD model of Spot

At a later stage we aborted the CAD model, because it lacked crucial details when compared with the configuration of the robot we had access to. Instead, we replaced the CAD model with a custom NeRF model captured using PolyCam[2], see fig 6a.

In fig 6b one can see clearly, that the learned model using the CAD model does not compare to the more detailed NeRF model. There are additional features and the point cloud more coherent, leading to better results.

Scene Configuration The set-up of the scene directly follows from imposed constraints. Spot is placed at the world origin and is static. The initial camera position is in close proximity to Spot and elevated. To sample coherent camera trajectories that resemble the motion of a human, BlenderProc includes a random walk sampler. Different scene backgrounds are randomly selected from the public 3D asset library Poly Haven.

3.2. Pose Estimation Algorithm

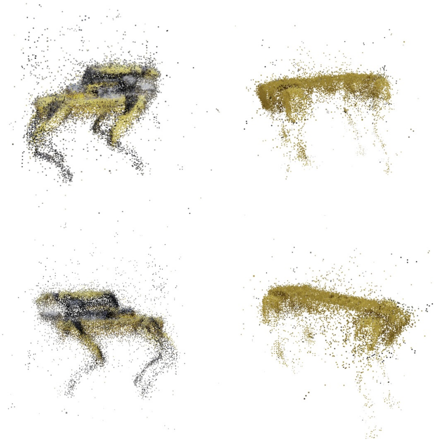
The authors of OnePose++ [4] graciously provided a tutorial detailing the procedure for running the pipeline with custom data[3]. We primarily adhered to this instructional resource. However, it should be acknowledged that the tutorial assumes the custom data is obtained using their

[1]https://github.com/heuristicus/spot_ros/tree/master

[2]see: https://github.com/PolyCam/polyform

[3]https://github.com/zju3dv/OnePose_Plus_Plus/blob/main/doc/demo.md

(a) NeRF model of Spot captured via PolyCam


(b) What OnePose++ learned. NeRF vs. CAD model

OnePose Cap Application. This app can be used to train OnePose on smaller custom objects. Spot proved to be too difficult, thus we were unable to utilize it. Consequently, we had to replicate the required data format for the OnePose++ pipeline independently, without reliance on the app.

After careful examination of the provided custom data example, we identified the specific files essential for implementing the OnePose++ pipeline for arbitrary objects. Each frame should be annotated with a rough bounding box that indicates the pose of the object. The initial bounding box has to be roughly estimated by tuning the dimension parameters manually using Blender and the robot model.

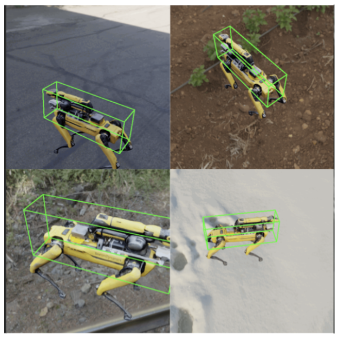


Figure 7. Synthetic images annotated with 3D bounding boxes to Spot

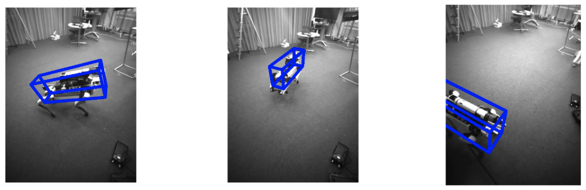
Initially, we aspired to generate the data format needed by the original code using the synthetic data described in Sec. 3.1.2. However, this approach proved unsuccessful as we lacked sufficient information about the OnePose Cap application, specifically the required. Box.txt file. Consequently, we opted to make minor modifications to the original OnePose++ code in order to accommodate the data formats we were capable of producing.

We trained the object detector using 8 different scenes, with each scene comprising 100 frames. Subsequently, we evaluated the trained model on real data comprising 1500 frames captured from HoloLens. The entire process was executed on a single GPU node (GeForce GTX TITAN X) and took approximately 1.5 hours.

4. Results

The ADD(S), cm-degree pose error and the Proj2D metrics are commonly used to evaluate the estimated object poses quantitatively. As stated earlier we could compute Spot's ground truth poses for the real-world data due to the difficulties arising in Sec. 3.1.1. Therefore, we visually assessed the frames in which pose estimation was successful. From the 1500 frames, 1000 frames (67%) were successfully annotated with the object poses. For some results, see fig 8a. Among the frames where pose-estimation was successful, Spot sometimes was also partially occluded and doing so-far unseen poses.

Generally, the pose estimation failed when the distance between the HoloLens and Spot was 'relatively' large, or the robot was aligned with the gaze direction, see Fig. 8b. These cases were not sufficiently covered in the synthetic dataset.


(a) Pose Estimation: Success Cases

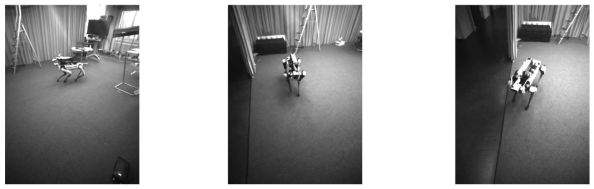

(b) Pose Estimation: Failure Cases

Figure 8. Pose Estimation Results

5. Discussion and Conclusion

In the presented work, we propose a modular pipeline to create synthetic data of the Spot robot in the CVG lab[4] using the standardized BOP format for object pose estimation. Furthermore, we adapted the existing OnePose++ [4] pipeline to use the synthetically created data for training a 6-DoF object pose estimation algorithm. Without having seen real-world data, the pre-trained network achieved promising pose estimation of the robot with monochrome data-streams recorded by the Microsoft HoloLens.

However, our experiments showed problems in detecting the robot if at a increased distance of the human observer or it the robot was aligned with the gaze direction i.e. the body was hardly visible, which can be observed in fig 8b. The limitations of our method arise from bias in the training data.

Unfortunately time did not suffice, to further investigate those artifacts but could be done in further work using the proposed pipeline for synthetic data generation.

Furthermore, it would be interesting to compare different pose estimation algorithms on this novel dataset.

6. Contributions of team members

We split the work up such that we were able to progress on multiple fronts even when we got stuck on specific challenges. For instance, when trying to obtain the correct projections from real-world data took longer than expected, we decided to generate synthetic data. Christopher's main focus was setting up the ROS environment and writing the necessary scripts for back-projection in Sec. 3.1.1. Michel took care of implementing the synthetic data pipeline while Mae and Jonas worked on getting different pose estimation algorithms running on their workstations and/or the Euler cluster. Mae mainly configured the OnePose++ [4] environment by modifying the original code to ensure successful execution with our created data and performed the necessary data format conversions. For more details, see this repository.

[4]http://cvg.ethz.ch

References

[1] Dengsheng Chen, Jun Li, Zheng Wang, and Kai Xu. Learning canonical shape space for category-level 6d object pose and size estimation, 2021. 2

[2] Wei Chen, Xi Jia, Hyung Jin Chang, Jinming Duan, Linlin Shen, and Ales Leonardis. Fs-net: Fast shape-based network for category-level 6d object pose estimation with decoupled rotation mechanism, 2021. 2

[3] Maximilian Denninger, Martin Sundermeyer, Dominik Winkelbauer, Youssef Zidan, Dmitry Olefir, Mohamad Elbadrawy, Ahsan Lodhi, and Harinandan Katam. Blenderproc, 2019. 1

[4] Xingyi He, Jiaming Sun, Yuang Wang, Di Huang, Hujun Bao, and Xiaowei Zhou. Onepose++: Keypoint-free one-shot object pose estimation without cad models, 2023. 1, 2, 3, 5

[5] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. volume 7724, 10 2012. 1

[6] Tomáš Hodaň, Frank Michel, Eric Brachmann, Wadim Kehl, Anders Glent Buch, Dirk Kraft, Bertram Drost, Joel Vidal, Stephan Ihrke, Xenophon Zabulis, Caner Sahin, Fabian Manhardt, Federico Tombari, Tae-Kyun Kim, Jiří Matas, and Carsten Rother. BOP: Benchmark for 6D object pose estimation. *European Conference on Computer Vision (ECCV)*, 2018. 1

[7] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization, 2016. 2

[8] Fu Li, Hao Yu, Ivan Shugurov, Benjamin Busam, Shaowu Yang, and Slobodan Ilic. Nerf-pose: A first-reconstruct-then-regress approach for weakly-supervised 6d object pose estimation, 2022. 2

[9] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, and Dieter Fox. DeepIM: Deep iterative matching for 6d pose estimation. *International Journal of Computer Vision*, 128(3):657–678, nov 2019. 2

[10] Yuan Liu, Yilin Wen, Sida Peng, Cheng Lin, Xiaoxiao Long, Taku Komura, and Wenping Wang. Gen6d: Generalizable model-free 6-dof object pose estimation from rgb images, 2023. 2

[11] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, nov 2004. 1

[12] Fabian Manhardt, Wadim Kehl, Nassir Navab, and Federico Tombari. Deep model-based 6d pose refinement in rgb, 2018. 2

[13] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020. 1

[14] Sida Peng, Yuan Liu, Qixing Huang, Hujun Bao, and Xiaowei Zhou. Pvnet: Pixel-wise voting network for 6dof pose estimation, 2018. 2

[15] T. Schneider, M. T. Dymczyk, M. Fehr, K. Egger, S. Lynen, I. Gilitschenski, and R. Siegwart. maplab: An open framework for research in visual-inertial mapping and localization. *IEEE Robotics and Automation Letters*, 2018. 2, 3

[16] Jiaming Sun, Zihao Wang, Siyu Zhang, Xingyi He, Hongcheng Zhao, Guofeng Zhang, and Xiaowei Zhou. Onepose: One-shot object pose estimation without cad models, 2022. 2

[17] Meng Tian, Marcelo H Ang Jr au2, and Gim Hee Lee. Shape prior deformation for categorical 6d object pose and size estimation, 2020. 2

[18] Sergey Zakharov, Ivan Shugurov, and Slobodan Ilic. Dpod: 6d pose object detector and refiner, 2019. 2