Solving scale ambiguity in Monocular ORB-SLAM with object detection based landmark

Wei Li
University of Ottawa
wli206@uottawa.ca

Abstract

In a visual SLAM system, the Monocular scale is inherently not able to be extracted correctly. We propose an algorithm that solves scale ambiguity in Monocular SLAM system by estimating scale correction factor from semantic object recognition. The goal of scale correction is to find an upscale/downscale factor that corrects the Monocular trajectory or map into the real scale. In our approach, MapPoints are assigned semantically to objects detected by a Mask R-CNN detector. Then scale representation to each object is calculated. We construct a scale library from RGB-D/ Stereo camera SLAM and then query the scale information to calculate the correction factor in Monocular SLAM. We implemented the system in ORB SLAM framework and demonstrate that the whole system is capable to reduce scale-induced error in Monocular SLAM. The source codes are provided in SourceCode

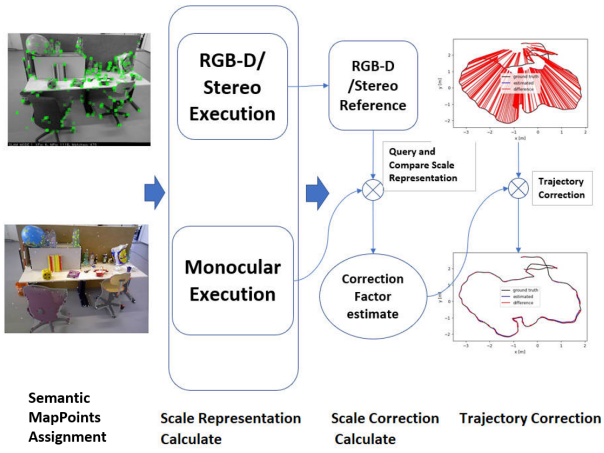


Figure 1. Absolute Scale estimation for Monocular ORB SLAM from semantic MapPoints assignment. Experiments in TUM RGB-D benchmark shows effective reduction to scale error.

1. Introduction

Visual Simultaneous Localization and Mapping (vSLAM) has been an actively researched and widely applied field in computer vision and Robotics. A SLAM system aims to estimate the position and orientation of sensors in an unknown environment, and simultaneously map the environment with a sparse representation.

The design of a SLAM system can vary significantly using different sensors or a mix of them. Among these sensors, visual SLAM, which primarily uses camera sensors has been intensely researched due to cameras' increasing ubiquity and cheap price.

A typical visual SLAM algorithm takes the estimation of the camera pose as the main goal and reconstructs the 3D map through the multi-view geometry. All 3D information is extracted from a series of photographs captured by cameras. Among visual SLAM algorithms, ORB-SLAM [12] [13] is the current state-of-the-art visual SLAM algorithm. Building on a series of successful applications [17] [9] [21] [20] [11], ORB-SLAM achieves real-time process using a standard modern multicore CPU.

While ORB-SLAM successfully solved Monocular camera SLAM, stereo camera SLAM and RGBD SLAM with high accuracy, the absolute scale in the monocular case remains ambiguous due to the fact that depth information, inherently, cannot be retrieved from monocular cameras. Without correct scale information, the scale of reconstructed trajectory and map can be arbitrary, typically manually assigned during the map initialization part of ORB-SLAM. Lacking a correct scale, the Monocular ORB-SLAM cannot work on scale-sensitive systems or applications that require true scales. Also, due to scale error, monocular SLAM cannot achieve an accuracy as low as its RGBD/Stereo counterpart before any calibration.

Traditionally, correct scale in Monocular SLAM is achieved by placing landmarks in the scene and measure-compare the scale of those landmarks. The method requires extra effort to physically place landmarks in the scene, and the method does not work when the environment is not ac-

cessible.

Inspired by the recent achievement in learning-based object detection, we aim to estimate the correct Monocular SLAM scale through exploiting semantic information in the scene – through objects.

The motivation is intuitive. We use a Mask R-CNN [7] object detector to constantly detect objects from consecutive frames. Then, similar to the 3D object reconstruction task, we assign MapPoints generated from the ORB-SLAM to instances of objects, based on the object masks provided by Mask R-CNN. Ideally, in each object, the reassigned map points should be able to describe the surface of the object. Furthermore, scale information can be extracted from the 3D point clouds of objects.

We design and implement an algorithm that integrates Mask R-CNN in ORB-SLAM and applies object-based map points assignment. Then, we design and apply a scale representation factor to each object (MapPoints cloud) extracted from the last step, and then use all objects' scale representation factors to infer the scale correction factor for the whole map/trajectory.

To obtain correct reference object scales, we develop the same system for RGB-D/stereo ORB-SLAM. After running the system in RGB-D/stereo mode for several sequences and saving seen object scale factors in a database, we calculated a reference object scale factor for each object category by taking a weighted average of scale representation factors in the database. Then we use the reference scale in Monocular mode to calculate the estimated upscale/downscale correction factor. Limiting the experiment to indoor, office scenes, the method can generally reduce scale error caused by Monocular scale ambiguity in TUM RGB-D dataset and the system can be transferred across several TUM sequences.

We review related work in Section 2, and then give a full presentation to the system in Section 3. Section 4 presents the results and analysis of the experiments. Finally, we conclude the paper with a discussion in Section 5.

2. Related works

We target the problem of absolute scale estimation in visual Monocular SLAM, specifically, the ORB-SLAM [12] [13]. ORB-SLAM provides a mature solution, without any assumption to the observed scene, to visual Monocular SLAM. With an open-source framework provided, ORB-SLAM is a benchmark in visual /Monocular SLAM problems.

There are many pieces of research that try to solve absolute scale in monocular visual SLAM. One way is to use additional sensors like IMU, GPS, LiDAR or stereo/RGB-D cameras. These sensors can provide enough information to reconstruct accurate camera position [2] or can infer map scale directly from depth information (ie. Kinect [14]) or displacements [16]. However, these methods are limited inherently because they require extra sensors which increase the cost to the system, also demanding extra computation and complexity for calibration of different sensors.

As the opposite, methods that evade the use of additional sensors explore the geometric or semantic information of RGB images. Most of the classical ways [3] [12] fix scale obscurity by utilizing ad-hoc information such as known scales (landmarks) or known motions in the initialization phase, and reconstructing the correct scale in the initial map. However, scale drift still occurs during tracking. In such cases, loop detection and closure techniques are often needed. The requirement of ad-hoc landmarks and/or motion also makes the solution less applicable for online usage or applications in inaccessible areas.

Still, using ad-hoc information exploits more geometric but not semantic information from the environment. On the other hand, many applications are also interested in semantic information such as object recognition. The semantic information can in turn helps to solve scale ambiguity. With the recent advance of object recognition and semantic segmentation using neural networks [19] [25] [8] [7] [1] [23], many successful object recognition networks have been incorporated with SLAM systems.

[22] proposed a Bayesian framework to infer the scale of objects detected by a YOLO object detector. The idea of the Bayesian inference framework is, although detected objects are uncertain in scale because the depth of object location cannot be retrieved, a robot should be able to infer the global scale of the environment by fusing the uncertainty from multiple objects. QuadricSLAM [15] is another work that uses object detection in object-oriented SLAM. The paper represents objects' scale properties as Quadric (i.e. surfaces such as ellipsoids). They show that Quadrics is a compact representation and can be conveniently computed from objects' bounding boxes. They derived a SLAM formulation that jointly estimates quadric and sensor pose. CubeSLAM [24] goes a step further to estimate object 3D cuboid bounding box for both static and dynamic SLAM. The framework generates high-quality cuboid proposals from 2D bounding boxes, and jointly optimizes camera poses, object 3D cuboid and points. They show that scale constraints gained from 3D objects cuboids improve camera pose estimation and reduce scale drift.

Another novel way is to infer scale-related information directly from deep neural network structures. [5] presents a Convolutional Neural Network that infers camera motion (i.e the speed of the camera) given successive monocular RGB frames. The speed estimates are used as regularizers in bundle adjustment, which corrects the scale drift without the need for explicit higher-level representations of the map to resolve scale. In [10], a Convolutional Neural Network with an encoder-decoder structure was trained to estimate a

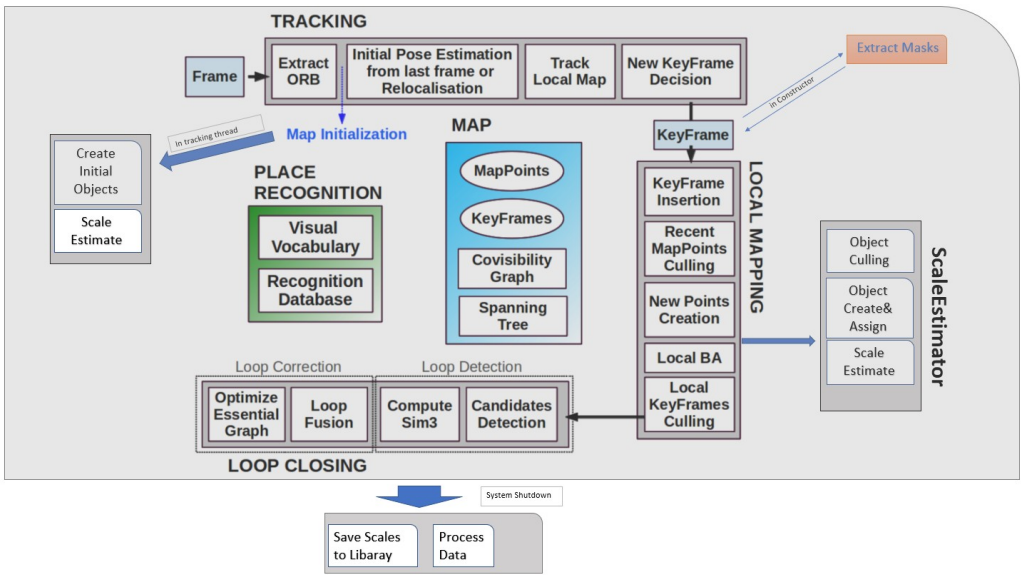


Figure 2. **Outline –** The pipeline of the propose scale correcting SLAM system

depth map given an RGB image. The obtained depth estimation is integrated into the ORB-SLAM framework [12]. [18] works on the base of [5], with the contribution of modifying the network architecture to fix several issues of [5], including loss of accuracy when a camera turns, and it improved the robustness against different sensor/environment configurations. They also show that models trained on synthetic driving data such as CARLA [4] yields comparable results to models trained on real driving data (KITTI [6]).

3. Method

In this section, we present the system to correct Monocular SLAM scale with semantic information obtained through Mask R-CNN. As illustrated in Figure 2, the system is built upon the original ORB-SLAM system with additional scale estimation functionalities. We give a brief over of the original ORB-SLAM system in 3.1 and introduce the Scale Estimator in 3.2.

3.1. ORB-SLAM Overview

The main goal of ORB-SLAM is to localize the sensor while building a map of the environment through visual features. The system uses ORB feature [17] in both localization and mapping. We call points extracted from ORB as **KeyPoints**. The points that work as the map to the environment are called **MapPoints**, which are built from KeyPoints through triangulation.

ORB-SLAM runs on three threads: tracking, local mapping and loop closing. The three threads work as the central part of the system and are connected through **KeyFrames**. KeyFrame is a subset of input frames with the main goal to reduce computation. Sufficient visual changes are needed to establish two consecutive KeyFrames. Also, graph structures are established among KeyFrames. Two structures are defined in ORB-SLAM:

- Covisibility Graph: with KeyFrames being nodes in the graph, two KeyFrames are connected by weight k if they share k common MapPoints
- Essential Graph: an subgraph of Covisibility graph where only nodes with weights exceeding a threshold are kept

Given a series of frames, the pipeline of ORB-SLAM always starts with estimating a frame's pose through extracting KeyPoints from the frame and matching them with KeyPoints of the last frame. However, an initial map and an initial KeyFrame are needed to start the whole pipeline. The process is called Map Initialization.

3.1.1 Map Initialization

Map Initialization runs on the tracking thread when there is no map already initialized.

For RGB-D and Stereo cameras, the initial map can be directly built from triangulating KeyPoints of the first left-right frames (stereo) or from the depth map (RGB-D). After the initial map is initialized, the first frame is decided as the initial KeyFrame.

For Monocular camera, the system needs to find two frames with sufficient ORB feature correspondences to reconstruct a relative pose of them, and then use the information above to triangulate the initial map. The two frames are

selected with high standard as the quality of the initial map significantly impact the quality of the SLAM result.

Two geometric transformation models, Homography and Fundamental matrix, are considered to estimate the relative pose of two initial frames, and the one with smaller reprojection error is selected. Given the relative pose of the two initial frames, 3D MapPoints are created from triangulation, and then full bundle adjustment is performed. The two frames are also selected as KeyFrames.

3.1.2 Tracking

Tracking is the main thread of ORB-SLAM, which processes every video frame. The tracking pipeline includes:

- **Extract ORB features**
- **Initial pose estimation** from either the previous Frame or through a Global Relocalization if the tracking is lost.
- **Track Local Map** refine the initial pose through reproject MapPoints in a **local map** of current frames and find more KeyPoints-MapPoints correspondences. The local map contains 1) a set of keyframes that share MapPoints with the current frame, 2) keyframes in the covisibility graph.
- **New KeyFrame Decision** decide if the current frame contains sufficient visual change to become a new KeyFrame. The KeyFrames are spawned generously and are culled strictly in the Local Mapping thread.

3.1.3 Local Mapping

Local Mapping work on a thread independent of the Tracking or Loop Closing thread, with the main function to create new MapPoints. Local Mapping only processes newly created KeyFrames, the pipeline of Local Mapping includes:

- **KeyFrame Insertion** update database of KeyFrame and related structures
- **Recent MapPoints Culling** perform a strict culling test to discard newly created MapPoints that are ambiguously or wrongly created.
- **New MapPoints Creation** creates new MapPoints by searching unmatched KeyPoints pairs in connected KeyFrames in the Covisibility graph. MapPoints are created from triangulating these KeyPoints pairs.
- **Local BA** optimize current KeyFrame, connected KeyFrames in Covisibility Graph, and all MapPoints observed by these KeyFrames.
- **Local KeyFrame Culling** detects and deletes redundant KeyFrames so that the BA complexity is controlled under longer-term running.

3.1.4 Loop Closing

Similar to Local Mapping, Loop Closing works on an independent thread and processes KeyFrames only after Local Mapping. Loop Closing detects loops, conduct loop fusion and closes the loop by performing pose graph optimization over the essential graph.

- **Loop Detection** queries bag of words representation of the current keyFrame in the recognition database and retain result KeyFrames that meet the Loop Detection requirements
- **Compute Similarity Transformation** between current KeyFrame and the loop candidates. Also, RANSAC iterations are performed on each candidate where candidates supported with enough inliers are accepted.
- **Loop Fusion** fuse duplicated MapPoints, and update KeyFrames graphs that are involved in the fusion.
- **Essential Graph Optimization** perform pose graph optimization over Essential Graph.

3.2. Scale Estimation Module

We give a system overview to the whole Scale Estimation Module in 3.2.1, then the core functionalities in the Scale Estimation Thread are introduced in 3.2.2.

3.2.1 System Overview

The idea to fix Monocular scale ambiguity is straightforward. Absolute scale can be retrieved from a depth map or through the stereo normal case in either stereo or RGBD ORB-SLAM system. Although Monocular SLAM has no access to this information, we can export semantic visual information from historical RGB-D/Stereo camera runs and build an external reference database for Monocular to use.

By building a system that identifies objects' masks in runtime, we dynamically assign new map points to identified object instances. From each object, a unified scale representation factor is calculated. The system work in the same way in either RGB-D/Steroe camera or Monocular camera mode. We build a database that stores these scale representation factors collected from the historic RGB-D/Stereo camera running. Then, a library of reference scale to each object class is calculated. The object scale database would effectively serve as our landmarks database to the Monocular SLAM.

Returning to the monocular ORB-SLAM case. As the same object detection functionality is also integrated with Monocular, when the Monocular system runs a new sequence and detected similar objects, the system queries the reference object's scale and compares the detected

object scales with the reference scales. If there is a scale discrepancy across multiple objects, the system calculates the scale correction factor which can upscale/downscale the trajectory or map such that they align back to the true scale.

Mask Extractor

The Semantic Segmentation module is the key to the scale reconstruction system. For one thing, we need an object detection model that is capable to detect objects at the instance level. During a SLAM running, all objects' instances detected by the model provide some inference to the scene's scale. For another thing, given that there is a possibility of partial overlapping object instances, the model should be able to generate object masks so that the front and behind relation (a rough depth) can be approximately retrieved.

We choose the mask R-CNN [7] trained on COCO dataset as the object extractor. The model can detect 80 object categories and provide object masks at the instance level.

Mask R-CNN can run at about 200ms per frame on a GPU. To further reduce the computation, we want to detect objects from KeyFrames instead of all frames. The reason is that the scale estimator reconstructs objects and scales from MapPoints, and Local Mapping (processing only KeyFrame) is the only part of ORB-SLAM where many MapPoints, potentially being parts of an object, are created.

In the implementation, the mask extractor is designed such that the GPU unit are signaled to extract object mask in the tracking thread whenever a new KeyFrame is decided and before the Local Mapping processing the KeyFrame. Furthermore, the system takes a retained RGB frame instead of a Gray Scale frame since the Mask R-CNN model is trained on the RGB dataset. The extraction model is run on an Nvidia GTX1660 backend through OpenCV 4.0 API. The whole system runs in real-time at the same rate as the ORB SLAM in the TUM RGB-D benchmark.

Initial Objects

Like localizing requires an initial map to start, the objects-MapPoints assignment algorithm requires initial objects created from the initial map. The algorithm is a simpler variation to the Algorithm 1 discussed in 3.2.2.B, excluding the need to search Mask-Object association. However, the computation of initial objects is implemented on the map initialization phase in the Tracking Thread just after the initial map is created. Optionally, the system can already approximately correct the map scale before SLAM even begin with initial objects.

Scale Estimator Thread

Scale Estimator is the main functioning module of the Scale estimation SLAM. The part runs in an independent thread called by Local Mapping thread just after the end of MapPoints creation. The reason behind using a new thread is to balance the computation. Experiments show that adding extra object-related computation to the three preexisting computing threads (Tracking, Local Mapping and Loop Closing) disrupts the multi-threading SLAM system and greatly increases the possibility of track losing or system freezing.

With the Scale Estimator thread, the scale correcting ORB-SLAM runs in total 4 computing threads and a GPU backend, which runs in real-time in standard CPU+GPU configuration.

Scale Estimator process KeyFrames through 4 processes: Object Culling, Object Create & Assign and Scale Estimate. The Scale Estimator thread will be elaborated in 3.2.2.

Library Builder and Monocular Reference

ORB SLAM works on three types of sensors: Monocular camera, RGB-D camera and stereo camera. While RGB-D and stereo cameras have different input data, the depth map of RGB-D and the left-right frames of stereo camera can be converted to one and the other easily. As a result, the two sensors are treated as the same in the ORB SLAM system with only differences in the input data processing. We keep this property so that the system works on both RGB-D and Stereo cameras with the same workflow.

In our system, RGB-D and stereo sequences are treated as the guide to the reconstruction of the Monocular scale, which we refer to as the Library Builder.

The idea of Library Builder is simple. The Object ORB-SLAM system is run in either RGB-D or Stereo camera mode over a variety of sequences. During the execution, the system collects a variety of object instances, along with their scales encoded in a Scale Representation Factor. After the execution, all object instances and their Scale Representation Factors are exported to an external scale library. The library logs all past seen object instances and their representation factors, and also categorize instances based on the object class in MS COCO format.

For each object category, a Reference Scale Representation Factor is calculated based on a weighted average Scale Representation Factors over all seen instances. In real, instantiate objects in each object class may vary in their scales even though they share the same class. However, it is possible to stabilize the average object scales with sufficient observation within sequences that come from the same environment, assuming that the scale representations are correctly extracted by the system.

Intuitively, an object instance with more surface MapPoints can provide a more confident estimation to its scale. Based on the assumption, we calculate the Reference Scale Representation Factor to each object category based on the average Scale Representation Factor weighted on instances'

number of MapPoints. Concretely, given a set of observed object instances $\delta_j^i \in \Delta^i$, where i is the class id and j is the instance id within the class. Define Ψ_j^i as the set of map points that consist of the point cloud of δ_j^i. $P(\delta_j^i)$ is the scale representation factor of the object δ_j^i. The reference scale representation factor R^i of object class i can be calculated as:

$$R^i = \frac{1}{\sum_{\delta_j^i \in \Delta^i} |\Psi_j^i|} \sum_{\delta_j^i \in \Delta^i} P(\delta_j^i) \times |\Psi_j^i|$$

Lastly, in the Monocular SLAM (reference mode), the external reference scale to each object class is queried for each detected object. The detected objects' scales in Monocular mode are calculated and compared with the reference scales. Then, a scale correction factor can be calculated from a model based on the quotient of the two scale factors. More details are described in Section 3.2.2.C.

3.2.2 Scale Estimator

The pipeline of Scale Estimator thread includes three functions: Object Culling, Object Creation & Assign and Scale Estimation.

A. Object Culling

An object δ_j^i in the Object SLAM system is defined by the class id i and the object's instance id j. In addition, a set of MapPoints Ψ_j^i belongs to the object δ_j^i.

The Scale Estimator starts with culling recently created objects that were wrongly assigned. The target is two or more different object instances that have a high percentage of overlapping MapPoints, and also share the same class id i.

When assigning MapPoints to objects with Algorithm 1, it is possible that MapPoints falls in two or more object masks that are overlapped, and thus being assigned to different object instances simultaneously. This happens either when two objects are actually overlapped, or when the Mask Extractor errorly created multiple masks for the same object. The second case is frequently observed in settings where the Mask R-CNN confidence threshold is set relatively low or is not tuned well for the scene. The Object Culling aims to solve the issue of the second case when the confidence threshold is not properly set.

To conduct Object Culling, we search object instances with more than $\alpha\%$ MapPoints also belongs to other objects (one or more), and then starting with the smallest object (by MapPoints number), we merge the smaller object to the larger overlapping object if $\beta\%$ of the smaller object's MapPoints also belongs to the larger object and the two objects share the same object class. Then we iteratively search such objects and merge them until no object that satisfies the α, β and the same-object-class constraint can be found. The Object Culling function successfully solves errorly overlapping mask problem when Mask R-CNN parameters are not properly set. In later experiments, we disabled the function when setting a high Mask R-CNN classification threshold which extracts mask only with above 90% classification confidence.

B. Object Creation and Assign

The second process in Scale Estimator is Object Creation and Assign. The process either assigns MapPoints without assignment to objects or creates new objects with these MapPoints. The pseudocode of the Object Creation and Assign process is described in Algorithm 1.

Given a new KeyFrame with a set of object masks extracted from the frame, the algorithm tries to associate new Object Masks with existing Objects. If there is no such association. the algorithm creates new objects from these Masks and unused MapPoints in the mask. To do so, the algorithm maintains an ObjectMask to Object association, and for each association there is a MapPoints counter that supports the association.

Firstly, the algorithm iterate over all assigned MapPoints to build the ObjectMask-Object association: whenever an assigned MapPoint falls in an ObjectMask, we add the association of the ObjectMask and the Object (to which the MapPoints belongs) by one. After the association is built, we pick the best-associated object to each Mask by picking for each mask the object with the largest number of supporting MapPoints. Then, for each unassigned new MapPoints, if the MapPoint falls in an Object mask with associated Object, the MapPoint will be assigned to the object, otherwise, a new object will be created with the MapPoint.

C. Scale Estimate

The objective of Scale Estimator is to estimate Scale Representation Factors to all detected Objects, both in Monocular mode and Stereo/RGB-D mode. The Scale Representation Factor is a float number and is extracted from 3D location information of the MapPoints in the Object.

Each MapPoint in SLAM has a 3D coordination with an origin defined in the Map Initialization phase. To define the Scale Representation Factor, we consider the center of mass of each Object's point cloud.

Let δ_j^i be an object instance of class i, and Ψ_j^i be the point cloud of object δ_j^i. Each MapPoint $\psi_k \in \Psi_j^i$ is a 3D vector in $\mathbb{R}^3$. The center of mass C can be calculated as

$$C = \frac{\sum_{\psi_k \in \Psi_j^i} \psi_k}{|\Psi_j^i|}$$

Then, we calculate the scale representation factor $P(\delta_j^i)$

Algorithm 1: ProcessKeyFrame(KeyFrame)

Result: Assigned MapPoints and new Objects

set of *objectMasks* are extracted with R-CNN ;
create empty Mapping of ObjectMask-Object pair to integer:
$Associations = \langle ObjectMask : Object \rangle : int$
,initialize with 0s ;
for *KeyFrame observed MapPoints* **do**
  if *MapPoint is assigned* **then**
    get mapPoint's corresponding *KeyPoint* in the KeyFrame ;
    if *KeyPoint fall in an ObjectMask* **then**
      for *all Object the MapPoint belongs to* **do**
        add entry $\langle ObjectMask : Object \rangle$ by 1 ;
      end
    end
  end
end
Create $BestAssociations = \langle ObjectMask : Object \rangle$ from $Associations$ by finding for each *ObjectMask* the *Object* with highest count
for *KeyFrame observed MapPoints* **do**
  if *MapPoint is NOT assigned* **then**
    get mapPoint's corresponding *KeyPoint* in the KeyFrame ;
    if *KeyPoints fall in an ObjectMask* **then**
      if *the ObjectMask has not been used to create object* **then**
        create new $Object = \langle class, instanceNum, MapPoints \rangle$ with one MapPoint ;
      end
      else
        assign the MapPoint to $Object$ according to the *BestAssociation* ;
      end
    end
  end
end

of object δ_j^i as the average L2 distance from MapPoints to the center of mass:

$$P(\delta_j^i) = \frac{\sum_{\psi_k \in \Psi_j^i} ||\psi_k - C||_2}{|\Psi_j^i|}$$

For Monocular SLAM, we want to find a Correction Factor that can adjust the scene (trajectory and map) scale so that it approximates the groundtruth scale. Ideally, the correction factor is a real number λ that after being multiplied to the observed Monocular trajectory, its difference (RMSE or absolute difference) to either the groundtruth trajectory or the RGB-D/Stereo observed trajectory is minimized.

The estimation of Correction Factor can be modeled by the quotient of detected monocular Object Scale Representation Factors and the external Reference Scale Representation Factors of the same class, which is denoted as R^i. We consider three weighted models to estimate the correction factor, the first model calculates the non-weighted mean of the quotient, the second and the third compute weighted mean of the quotient and take into account either the number of MapPoints in the object, or both the number of MapPoints and the number of observation in scale database Δ^i.

For each monocular run, there is a set of Object Instances Ω created in Object Creation & Assign. For each instance $\delta_j^i \in \Omega$, i is the object class of δ_j^i, $P(\delta_j^i)$ is the estimated scale of δ_j^i. Ψ_j^i is the point cloud of MapPoints. Also, we denote R^i as the reference scale of class i in the scale library. Δ^i is the set of all previously observed class i instances in the scale library, and $|\Delta^i|$ is the number of times instances of class i appear in the library. The models estimate a scale correction factor ϵ.

- **no weighted:** given Monocular Object Instances and scales, and the scale library, the estimated scale correction is
$$\epsilon = \frac{1}{|\Omega|} \sum_{\delta_j^i \in \Omega} \frac{R^i}{P(\delta_j^i)}$$
- **weight #MP and #Observations:** given Monocular Object Instances' scales and the scale library, the estimated scale correction weight on both how frequently an instance of object class i appears in the scale library, and how many MapPoints the object instance has.
$$\epsilon = \frac{1}{\sum_{\delta_j^i \in \Omega} wmp(\delta_j^i)} \sum_{\delta_j^i \in \Omega} \frac{R^i}{P(\delta_j^i)} \times wmp(\delta_j^i)$$
with
$$wmp(\delta_j^i) = \frac{|\Psi_j^i| \times |\Delta^i|}{\sum_{k \in C(\Omega)} |\Delta^k|}$$
$C(\Omega)$ is set of unique object class appear in Ω
- **weight #MP** given Monocular Object Instances' scales and the scale library, the estimated scale correction weight only on the number of MapPoints in object instances.
$$\epsilon = \frac{1}{\sum_{\delta_j^i \in \Omega} |\Psi_j^i|} \sum_{\delta_j^i \in \Omega} \frac{R^i}{P(\delta_j^i)} \times |\Psi_j^i|$$

D. Parameters

We discuss parameters in the system. There are six parameters in the system, two parameters are used by Mask Extractor, two are used by KeyFrame for Mask Culling and two are used by Object Culling in the Scale Estimator.

- Mask R-CNN Mask Extractor
  confThreshold: The mask is extracted only when the classifier in Mask R-CNN returns a score larger than the confThreshold.
  maskThreshold: threshold of possibility for a pixel to be classified as occupied instead of the background

- MaskCulling
  minSize: filter out small masks that have less than 1100 pixels. The hard coded minimal size of 1100 is intended to filter out very small areas that are unlikely to collect MapPoints or to reconstruct a sensible scale from
  maxSize: filter out very large masks of over 120000 pixels. Very large masks usually take half or more areas of a frame in TUM RGB-D dataset. The result is that the particularly big mask covers all other masks and disturbs the MapPoints assignment of these objects. Very large masks generally are in the categories such as 'table' or 'bench' in our experiment.

- Object Culling
  α **and** β : The two parameters are discussed in 3.2.2.A. They are set to 0.5 and 0.6 respectively in preliminary experiments.

Ablations to the confThreshold and maskThreshold are included in appendix A.

Experiment	Iteration	#run	
Single Sequence	each of 6 sequences	RGBD	50
		Monocular	30
Transferability	each of 4 sequences	RGBD	50
		Monocular	30
MaskExtract Ablation	each parameter setting	RGBD	30
		Monocular	20

Table 1. Summary of number of runs in each experiment, each sequences or parameter settings are run on either RGB-D or Monocular mode for the given times

4. Experiments

We test the system's ability to estimate scale correction factor from Scale Library, and the quality of the corrected trajectory. The benchmark to our comparison is either RGB-D trajectories or the groundtruth trajectories provided by the TUM RGB-D benchmark.

TUM RGB-D benchmark

We use TUM RGB-D benchmark in the experiments. Sequences in TUM RGB-D benchmark are shot from indoor office environment RGB-D camera with groundtruth camera trajectories. Since the dataset contains only RGB-D data, we build the Scale Library from RGB-D builder mode and reconstruct the scale in Monocular mode. The system is able to deal with Stereo camera input with minimal modification.

Sequences in TUM RGB-D dataset are recorded in three different sensors with different intrinsic parameters. Sequences are tagged by either freiburg1, freiburg2 or freiburg3. It is observed that the Monocular ORB-SLAM result that runs on freiburg3 has the largest scale discrepancy compared to the groundtruth, while the result from the other two categories is relative close to groundtruth.

We choose 6 sequences in the experiment from the TUM RGB-D dataset. The chosen sequences are relatively rich in the number of objects and the diversity of object types. That is includes: fr1_desk, fr1_room, fr1_xyz, fr2_xyz, fr3_long_office, fr3_teddy.

Performance Evaluation

For evaluation, we consider two types of indices. The first one is the quotient of the modeled correction factor and a correction factor comes from optimization. The second one is the error from aligning corrected monocular trajectories with the groundtruth trajectories.

We use evaluate_ate_scale to get a groundtruth scale correction factor. The script takes a TUM RGB-D trajectory from an arbitrary SLAM system and aligns it to the sequence's groundtruth trajectory such that the error between the two trajectories is minimized. For trajectories that are not absolute scaled like Monocular trajectories, it also produces a scale correction factor to correct the scale. We call the scale correction factor an Optimized Correction Factor. Then the estimated correction factor is compared with the Optimized Correction Factor by calculating the average quotient of the two factors.

$$\text{meanDiff} = \sqrt[n]{\prod_i D_i}$$

with

$$D_i = \frac{\text{est\_correction}}{\text{opt\_correction}}$$

with i from 1 to n iterating the number of monocular execution. Ideally, the meanDiff index should be as close to 1 as possible.

We also compare the groundtruth trajectory alignment RMSE of: original Monocular trajectory, the Monocular trajectory adjusted with our system, the optimized trajectory with evaluate_ate_scale, and the RGB-D trajectory.

Since the estimated scale comes from RGB-D reconstructed scale, generally the trajectory alignment error that comes from the scale correction system should not exceed the RGB-D trajectory alignment error.

Setup
All experiments are run offline. In the first step, a database of reference object scale is collected from a number of RGB-D runs, either from the same sequence(the single sequence experiment), or from different sequences(the transfer experiment). Then in the second step, we execute the scale reconstruct system in Monocular mode, taking the reference scale library constructed in the previous step as input, and estimating the scale correction factor during each execution. Since the system suffers from randomness from two sub-systems, the ORB-SLAM and the Mask R-CNN, we calculate the average results from a number of Monocular runs. The total number of runs in all experiments are given in Table 1.

The scale ORB-SLAM system kept the same setting in all experiments except the Mask R-CNN ablation in appendix A. The experiments are run with Mask R-CNN confThreshold 0.9, and maskThreshold 0.6. Extracted masks with a total number of pixels being smaller than 1100 or larger than 120000 are discarded. Furthermore, the Object Culling function is disabled.

Finally, for each detected object instance we take the last estimation of object scale representation factor a the scale sample. The sample is either added to the scale library in RGB-D/Stereo building mode or taken as input to calculate estimated scale correction factor in Monocular reference mode.

4.1. Compare Scale Estimation Models

Three weighted models of scale estimator are presented in 3.2.2.C. We compare the quality of the scale correction factor from the three models.

In the experiment, we use RGB-D trajectory scale as the target scale. For each Monocular run and RGB-D run, a scale correction factor γ that minimizes the Monocular trajectory and RGB-D trajectory difference is calculated through optimization. Then, for each monocular run, the average of γ over all RGB-D trajectories is calculated. The value is plotted as green dots in Figure 3. Since the scale of Monocular SLAM is arbitrarily set and is affected by many factors in a SLAM system, while RGB-D trajectory scale is relatively stable as they are very close to the groundtruth, the average of γ is often unique to each Monocular trajectory.

It is observed that the targeted scale correction factor γ has a considerable amount of variance with respect to different Monocular runs. This is consistent with the previous assumption about the Monocular initial scale. Consequently, the target of the system is not to find an absolute correction factor but to find a unique correction factor for each Monocular input. For better visualization, instead of presenting the absolute modeled correction factor, the absolute difference of the modeled scale correction factor and the target scale factor is calculated and plotted.

The modeled **no weighted**, **weighted #MP** and **weighted #appear and #MP** scale correction factors are calculated for each Monocular run. Then their absolute difference to the target scale correction factor is calculated and plotted as red, blue and black dots respectively. These absolute differences should ideally be minimized to 0.

Figure 3 shows the scale correction comparison results of three sequences. To evaluate the result, an ideal model should has both lower average differences and also be robust to the randomness of the quality to Monocular runs. It is observed **no weighted** model returns a worse correction factor in some circumstances. On the other hand, while the other two models' mean differences are close, the model **weight# MP and # appear** generally has a closer estimation to RGB-D scale and is relatively robust to Monocular scale quality.

The system returns scale correction factors close to the RGB-D in the three sequences. However, as the Monocular results from freiburg1 sequences are generally already close to RGB-D scale, estimated scale correction may not improve the original result. This is confirmed in the later experiments.

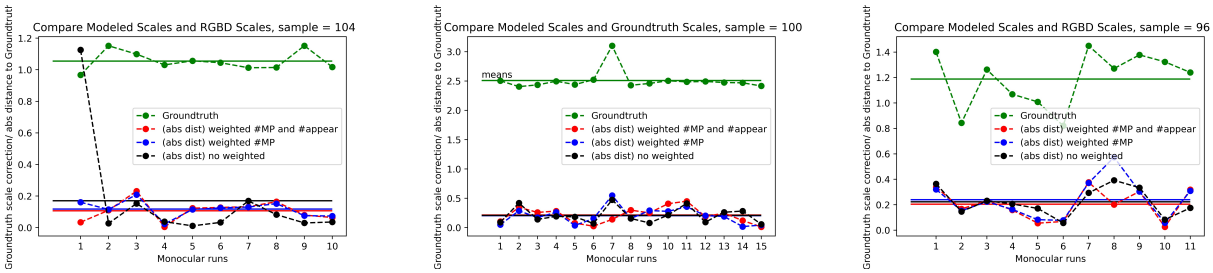
In the following experiment, the **weight# MP and # appear** model is used as the scale estimation model.

4.2. Single Sequence Performance

In the experiment, the scale estimation SLAM system collects a scale database exclusively from one sequence and reconstructs scene scale for the same sequence. Since it is the same sequence and most of the time the object detector picks up the same object, the randomness from the object detector and the scale variance in real objects are limited. It is an easier task for the system.

We align the Monocular trajectory corrected by the system with the groundtruth trajectory provided by the TUM RGB-D dataset. Then the average RMSE of the two aligned trajectories is calculated. For comparison, we also compute 1) the average RMSE in the same manner for RGB-D trajectory 2) Monocular RMSE corrected with an optimized correction factor calculated by evaluate_ate_scale. 3) RMSE of the original monocular trajectory fitting to the groundtruth with only translation and rotation. 4) the geometric mean of the quotient of estimate correction factor and optimal correction factor, which should ideally be 1.

We conduct the experiment in six TUM sequences with 50 RGB-D runs and 30 Monocular runs. Examples of the aligned trajectories are given in Appendix C. Table 2 shows



(a) fr1_desk (b) fr3_long_office (c) fr1_xyz

Figure 3. Compare Scale Estimation Models by distance to the RGB-D scale

the results of single sequence experiments.

Sequence	RMSE					Correction Difference
	Mono estimate	Mono origin	Improve	Mono opt	RGBD	
f1_desk	0.107	0.108	0.108	0.014	0.019	1.075
f1_room	0.386	0.596	0.353	0.191	0.054	1.011
f1_xyz	0.055	0.066	0.161	0.010	0.011	0.822
f2_xyz	0.052	0.068	0.236	0.010	0.011	0.850
f3_long_office	0.310	2.119	0.854	0.014	0.010	0.934
f3_teddy	0.128	0.231	0.446	0.038	0.014	1.536

Table 2. Single Sequence trajectories alignment accuracies and Correction Factor differences

We found that the system can generally improve Monocular trajectory align accuracy upon the origin Monocular trajectory. In fr3_long_office, the system reduces more than 85% RMSE from the original trajectory. The improvement is more prominent when the Monocular scale constructed by vanilla ORB SLAM is largely different from the groundtruth (for example, fr3_long_office and fr3_teddy).

Overall, it is hard to compare the result of different sequences. It is observed that the resulting error is determined by a variety of factors. We list some of the observations in the experiments, which also points out some of the limits of the system.

1) The effect of correction is not as good when the scale of the original Monocular trajectory is already close to the true scale. For example, in fr1_desk, even though the meanDiff of estimated scale correction and the optimized scale correction is close to 1, the improvement of alignment accuracy is marginal. In some cases, there is even worsen accuracy. This observation suggests that the scale correction method has an upper bound to the accuracy it can achieve, as the whole system is based on average landmarks from the scale database.

2) The original Monocular trajectory quality also affects the accuracy. The sequence fr1_room included a lot of fast camera movements. Vanilla ORB SLAM running in fr1_room experienced multiple tracks lost, resulting in a trajectory with compromised integrity. An incomplete trajectory makes the alignment harder even with an optimized scale, which also limited the result of the scale correction system.

3) It is observed that the Mask R-CNN fails to recognize some common objects frequently. Frequently mistaken object types include teddy bear and human, rubbish bin and cup, cabinet and refrigerator. These mistakes are generally similar in shape but have very different real scales. One way to solve the problem is to increase the confThreshold in the Mask R-CNN object detector.

4) Some sequences contain a very limited number of objects. For example, in sequence fr1_xyz and fr2_xyz only 3 to 4 object instances, on average, is detected by the Mask R-CNN model. With only 3 to 4 landmarks the system is more vulnerable to uncertainty from various parts of the system.

5) The MapPoint assignment algorithm does not handle deep depth well. In fr3_teddy, the camera scans a teddy bear sitting in the middle of the room from 360 degrees. The ORB SLAM picks up points from far beyond the teddy bear in the room. Some of these high depth points also were included in the detected teddy, making the estimated scale of the teddy bear bigger than the actual scale. This results in a higher meanDiff of 1.536.

4.3. Sequence transfer Performance

In this part, we test the scale SLAM system in four office sequences that share some common objects. Instead of running Monocular reference on the scale database built from the same sequence, we test the SLAM's ability to transfer between different sequences. In other words, we simulate real applications where video sequences that are processed have never been seen before.

Same as before, we run each sequence in RGB-D mode 50 times (to collect object scale database), and run each sequence in Monocular scale reference mode 30 times. We split the scale library into 2 parts. Firstly, for each target Monocular sequence, we build a scale library from RGB-D result of all other three sequences, i.e. exclude the target sequence. Secondly, we build a scale library from the results of all four sequences, i.e. include the target sequence. Then, we test the Monocular SLAM correction system using either the excluded reference scales or the included ref-

Database	Sequences	RMSE					Correction Difference
		Mono estimate	Mono origin	Improve	Mono opt	RGBD	
include	fr1 _desk	0.112	0.113	0.010	0.014	0.019	1.050
	fr1 _room	0.534	0.576	0.072	0.219	0.054	1.106
	fr1 _xyz	0.073	0.085	0.134	0.015	0.011	0.808
	fr3_long_office	0.192	2.069	0.907	0.015	0.010	0.967
exclude	fr1 _desk	0.119	0.113	-0.056	0.014	0.019	1.051
	fr1 _room	0.585	0.552	-0.059	0.204	0.054	1.030
	fr1 _xyz	0.074	0.085	0.131	0.015	0.011	0.808
	fr3_long_office	0.423	2.063	0.795	0.016	0.010	1.093

Table 3. Sequences Transfer trajectories alignment accuracies and Correction Factor differences

database	#object classes	total observation	model RMSE	correction Difference
excluded	11	938	0.423	1.093
one_seq	11	1157	0.310	0.934
included	12	2095	0.192	0.967

Table 4. Compare Scale reconstruction quality by scale database number of classes and number of instances

erence scale. The experiment was run on freiburg1_desk, freiburg1_room, freiburg1_xyz and freiburg3_long_office.

Table 3 shows the transfer RMSE and Correction Differences in the same format as the previous experiment. In both excluded database and included database experiments, the three fr_1 sequences maintains a correction difference close to 1 and the RMSE in the same level as the original result. The included scale databases are in general slightly better. It is also intuitive as including own observation will bias the average of the database toward the target sequence.

On the other hand, included database of the fr3_long_office actually obtains a better result than the single sequence experiment with 90.7% of improvement over the original Monocular trajectory. The difference between the two results is that the included database has more observed object instances in the database in addition to that is observed in fr3_long_office. We give a comparison of some object classes of the two scale databases in Appendix B. The result shows that the reference scales to each object class calculated from averaging all observations do not differ a lot in either the single sequence database or the transfer database.

Table 4 calculates the number of objects types and the number of total observed instances from the included database, single sequence database and excluded database of the sequence fr3_long_office. The result suggests that more observed object instances in the database may help the quality of scale estimation.

5. Conclusions

In this project, we present a method integrated with ORB SLAM system that estimates and reconstructs absolute scale in Monocular SLAM. The method uses a Deep-Learning-based semantic mask extractor and conducts the scale reference from historical RGB-D/Stereo data. The whole system is implemented with OpenCV 4 and runs in consumer-level CPU+GPU setup in real-time. From the experimental results in TUM RGB-D benchmark office sequences, conducted for both single sequence scale library and transferring sequence scale library, it was observed that the proposed method is effective to reduce the scale error caused by Monocular SLAM ambiguity. The system is still limited in several aspects including the Mask R-CNN extractor, depth points handling, and is also limited by the original ORB-SLAM performance. A future work to improve the system can focus on:

- use semantic extraction model that capable to detect wider range of objects with higher accuracy
- optimize the depth points handling mechanism
- optimize the scale estimation model, for example, using geometric mean weighted with the number of MapPoints and the number of instances.

References

[1] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: Real-time instance segmentation. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9156–9165, 2019. 2

[2] Xieyuanli Chen, Hui Bin Zhang, Huimin Lu, Junhao Xiao, Qihang Qiu, and Y. Li. Robust slam system based on monocular vision and lidar for robotic urban search and rescue. *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pages 41–47, 2017. 2

[3] A. Davison. Real-time simultaneous localisation and mapping with a single camera. *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 1403–1410 vol.2, 2003. 2

[4] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017. 3

[5] Duncan P. Frost, D. W. Murray, and V. Prisacariu. Using learning of speed to stabilize scale in monocular localization and mapping. *2017 International Conference on 3D Vision (3DV)*, pages 527–536, 2017. 2, 3

[6] Andreas Geiger, Philip Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012. 3

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask r-cnn. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42:386–397, 2020. 2, 5

[8] Zilong Huang, Xinggang Wang, Lichao Huang, Chang Huang, Yunchao Wei, Humphrey Shi, and Wenyu Liu. Ccnet: Criss-cross attention for semantic segmentation. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 603–612, 2019. 2

[9] Georg S. W. Klein and D. W. Murray. Parallel tracking and mapping for small ar workspaces. *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 225–234, 2007. 1

[10] Y. Li, Chenggang Xie, Huimin Lu, Xieyuanli Chen, Junhao Xiao, and H. Zhang. Scale-aware monocular slam based on convolutional neural network. *2018 IEEE International Conference on Information and Automation (ICIA)*, pages 51–56, 2018. 2

[11] Christopher Mei, Gabe Sibley, and P. Newman. Closing loops without places. *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3738–3744, 2010. 1

[12] Raul Mur-Artal, J. Montiel, and J. D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31:1147–1163, 2015. 1, 2, 3

[13] Raul Mur-Artal and J. D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33:1255–1262, 2017. 1, 2

[14] Richard A. Newcombe, S. Izadi, Otmar Hilliges, D. Molyneaux, David Kim, A. Davison, P. Kohli, J. Shotton, Steve Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, 2011. 2

[15] Lachlan Nicholson, Michael Milford, and Niko Sünderhauf. Quadricslam: Dual quadrics from object detections as landmarks in object-oriented slam. *IEEE Robotics and Automation Letters*, 4:1–8, 2019. 2

[16] Gabriel Nützi, S. Weiss, D. Scaramuzza, and R. Siegwart. Fusion of imu and vision for absolute scale estimation in monocular slam. *Journal of Intelligent & Robotic Systems*, 61:287–299, 2011. 2

[17] Ethan Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. *2011 International Conference on Computer Vision*, pages 2564–2571, 2011. 1, 3

[18] D. Rukhovich, Daniel Mouritzen, R. Kaestner, M. Rufli, and A. Velizhev. Estimation of absolute scale in monocular slam using synthetic data. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 803–812, 2019. 3

[19] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:640–651, 2017. 2

[20] H. Strasdat, A. Davison, J. Montiel, and K. Konolige. Double window optimisation for constant time visual slam. *2011 International Conference on Computer Vision*, pages 2352–2359, 2011. 1

[21] H. Strasdat, J. Montiel, and A. Davison. Scale drift-aware large scale monocular slam. In *Robotics: Science and Systems*, 2010. 1

[22] E. Sucar and J. Hayet. Bayesian scale estimation for monocular slam based on generic object detection for correcting scale drift. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–7, 2018. 2

[23] Xinlong Wang, Rufeng Zhang, Tao Kong, Lei Li, and Chunhua Shen. Solov2: Dynamic and fast instance segmentation. *arXiv: Computer Vision and Pattern Recognition*, 2020. 2

[24] Shichao Yang and S. Scherer. Cubeslam: Monocular 3-d object slam. *IEEE Transactions on Robotics*, 35:925–938, 2019. 2

[25] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6230–6239, 2017. 2

Appendix

Supplementary Material

A. Ablation: Mask and conf threshold

In this section, we conduct ablation on two parameters in the Scale Estimation SLAM system, the mask threshold and the confidence threshold in the mask extractor.

In the previous experiments, we set the mask threshold to 0.6 and the confidence threshold to 0.9. In this experiment, we either fix the mask threshold to 0.6 and adjust the confidence threshold or fix the confidence threshold to 0.9 and adjust the mask threshold. The objective of the experiment is to see how different Mask Extractor parameters pairs affect the final aligned RMSE of the whole system.

The setting of each parameter was run in RGB-D 30 times and was run in Monocular 20 times. Then we calculate the RMSE by aligning the corrected Monocular trajectories with the dataset groundtruth like experiments in 4.2 and 4.3. The result is presented in Table 5 and Table 6.

Firstly, the effect of changing the mask threshold can be directly observed from the visualizer of the system. Figure 4 shows different maskThreshold settings with confThreshold set to 0.9 in the same frame. With a smaller maskThreshold, the mask extractor extract object masks with larger areas. A larger mask may be helpful to cover more edges and boundaries of the object, and thus the model collects more points for the object. However, larger masks may also wrongly cover background object masks or feature points. For one thing, if the object culling function is enabled, two objects of the same class will be merged by error. For another thing, when the background point being covered is far away, the scale of the object will be exaggerated.

On the other hand, changing the confThreshold also impacts the quality of scale estimation. From our observation, lowering the confThreshold will result in more objects instance being detected, but generally with wrong classifications. Furthermore, objects that are not included in the MS COCO dataset class can be wrongly classified as objects in the dataset. For example, cabinets are classified as refrigerators, toy cranes are classified as sinks, rubbish bins are classified as cups.

	RMSE					Correction Difference
conf Threshold	Mono estimate	Mono origin	Improve	Mono opt	RGBD	
0.7	0.484	2.085	0.768	0.066	0.011	0.990
0.8	0.665	2.089	0.681	0.200	0.010	0.967
0.9	0.310	2.119	0.854	0.014	0.010	0.934

Table 5. Ablation result with fixed maskThreshold to 0.6

With a fixed maskThreshold, we test different confThreshold. It was observed that the quality of detected objects greatly deteriorated with confThreshold lower or equal to 0.6. We tested the value of 0.7, 0.8 and 0.9. In Table 5, the mask extractor with confThreshold set to 0.9 results in significantly better alignment error compare to the other two. With the specific Mask R-CNN model, avoiding detection with low quality can help the system with scale reconstruction.

In Table 6, we tested four different maskThreshold values of 0.2, 0.4, 0.6 and 0.8. Unlike confThreshold, the results are relatively close. Overall 0.6 maskThreshold provides a better balance between the benefit and drawback of small maskThreshold for the fr3_long_office sequence. The sequence contains lesser shots with ORB points in the far background and also contains lesser object overlapping. However, for better performance, the parameter may still need to be tuned for other environments.

	RMSE					Correction Difference
mask Threshold	Mono estimate	Mono origin	Improve	Mono opt	RGBD	
0.2	0.398	2.110	0.811	0.090	0.011	0.908
0.4	0.420	2.065	0.797	0.013	0.012	0.878
0.6	0.310	2.119	0.854	0.014	0.010	0.934
0.8	0.365	2.069	0.824	0.013	0.011	1.092

Table 6. Ablation result with fixed confThreshold to 0.9

B. Transfer Database

We present three common object types: chair, keyboard and tv from scale database that built from 1)fr3_long_office sequence 2) fr1_desk, fr1_room and fr1_xyz, i.e. the excluded transfer database of fr3_long_office.

In Figure 5, each point represents an object instance detected by the system in historical runs. Darker points (to the right of the x-axis) indicate that the sample point has more #MapPoints. Each point's y coordinate indicates its estimated scale from the model described in 3.2.2.C. The red line is an unweighted average scale of all objects' scales, while the green line is a weighted average scale based on the volume (number of MapPoints) in object instances, which is also what is used in the previous experiments. Although built from different sequences, the reference scales calculated from the two scale databases are in general close to each other.

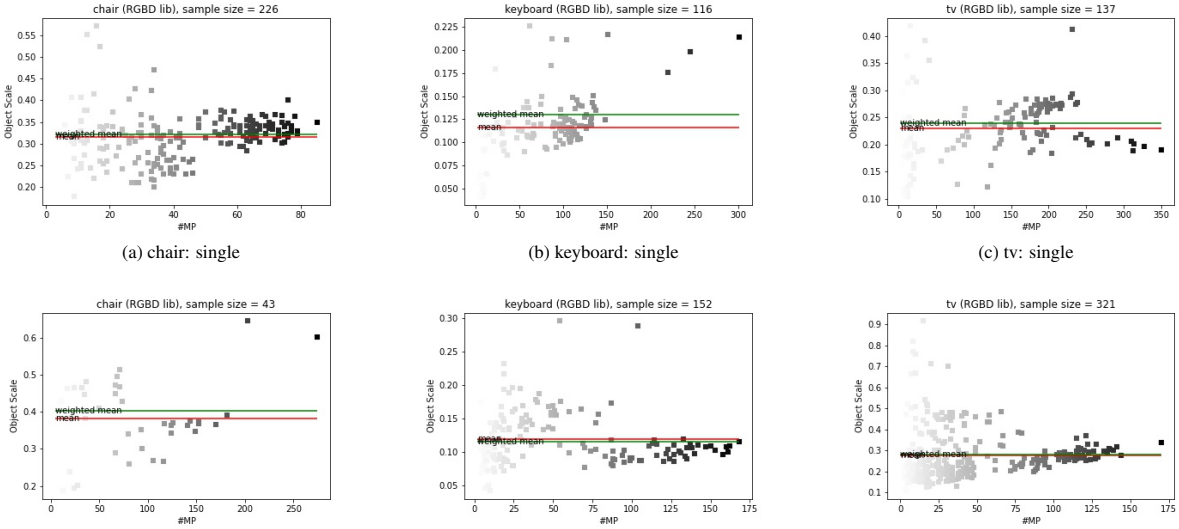
C. Example Trajectory Alignment

Sample aligned Monocular trajectories with the scale correction SLAM system (before and after) are presented

(a) maskThreshold:0.3 (b) maskThreshold:0.4 (c) maskThreshold:0.5

(d) maskThreshold:0.6 (e) maskThreshold:0.7 (f) maskThreshold:0.8

Figure 4. maskThreshold effect in the same frame

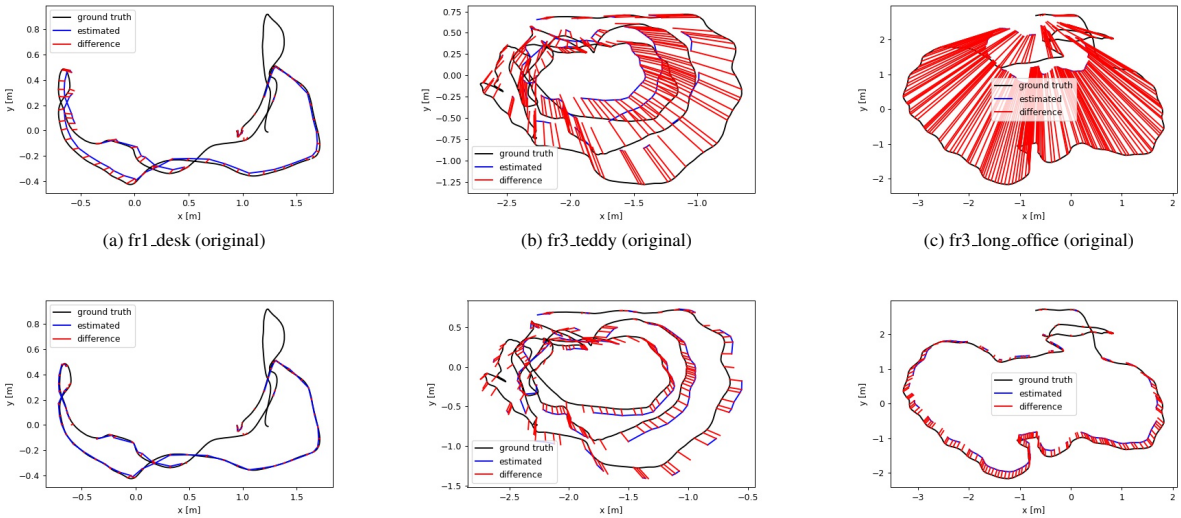


(a) chair: single (b) keyboard: single (c) tv: single

(d) chair: transfer(excluded) (e) keyboard: transfer(excluded) (f) tv: transfer(excluded)

Figure 5. Example Objects' Scale database comparison: single sequence database and transfer database

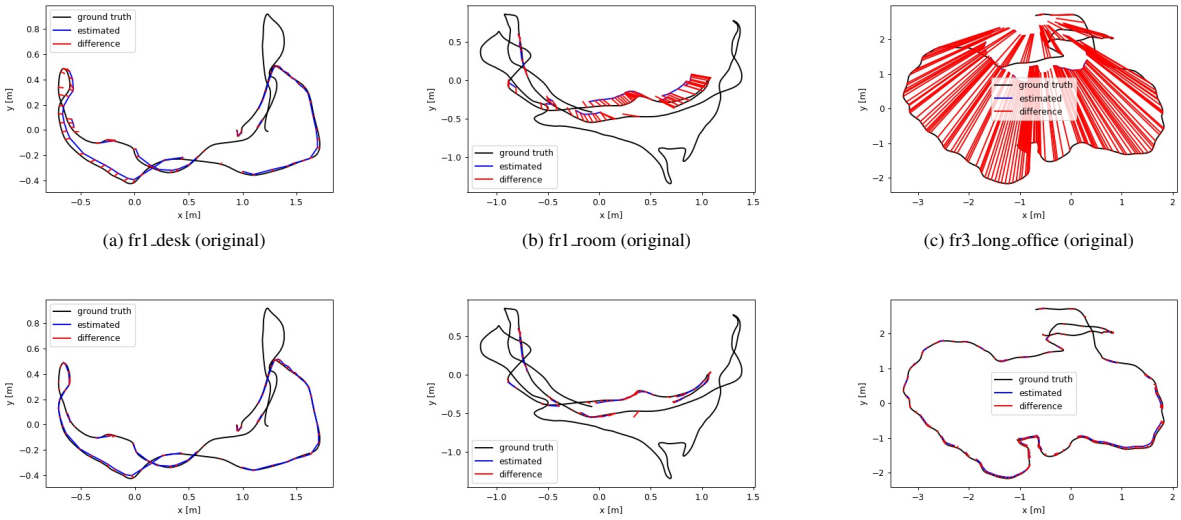
in Figure 6 and Figure 7. Both single sequence database and transfer database are capable to fix the scale ambiguity.



(a) fr1_desk (original) (b) fr3_teddy (original) (c) fr3_long_office (original)

(d) fr1_desk (corrected) (e) fr3_teddy (corrected) (f) fr3_long_office (corrected)

Figure 6. Single Sequence Alignment Samples



(a) fr1_desk (original) (b) fr1_room (original) (c) fr3_long_office (original)

(d) fr1_desk (corrected) (e) fr1_room (corrected) (f) fr3_long_office (corrected)

Figure 7. Transfer Sequence Alignment Samples