Deep Neural Networks for Robot Navigation Using Visual Teach and Repeat

by

Faraz Shamshirdar

B.Sc., Amirkabir University of Technology, 2016

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Science

© Faraz Shamshirdar 2019
SIMON FRASER UNIVERSITY
Summer 2019

Copyright in this work rests with the author. Please ensure that any reproduction or re-use is done in accordance with the relevant national copyright legislation.

Approval

Name:	**Faraz Shamshirdar**
Degree:	**Master of Science (Computing Science)**
Title:	**Deep Neural Networks for Robot Navigation Using Visual Teach and Repeat**
Examining Committee:	**Chair:** Angelica Lim Assistant Professor
	Richard T. Vaughan Senior Supervisor Professor
	Mo Chen Supervisor Assistant Professor
	Yasutaka Furukawa Internal Examiner Assistant Professor
Date Defended:	**May 27, 2019**

Abstract

Navigation is an integral component of any autonomous mobile robotic system. Typical approaches to navigation consist of metric localization based on different sensory information and a plan to reach a metric target. Visual navigation is an instance of navigation utilizing only visual information for localization. This thesis demonstrates 1) Sparse semantic object detections as robust features for visual teach and repeat; 2) Deep convolutional neural network (CNN) models to extract high-level features for visual teach and repeat; 3) Visual localization-space trails for multi-agent navigation.

The first application demonstrates an autonomous unmanned aerial vehicle (UAV) equipped with a monocular camera and capable of repeating a taught path using only sparse semantic object detections. This method employs a pre-trained deep model to detect semantic objects. The model is able to detect objects in a video at frame rate reliably. We show that semantic objects are sufficient to be used as landmarks for visual teach and repeat; they are repeatable, highly invariant to different lighting condition and surface appearance changes.

The second application is an extension to the first application and proposes deep CNN features for visual teach and repeat. This method utilizes salient features generated by a deep model such as textures and patterns over only semantic features. Two deep neural networks are used in this system: 1) A pre-trained model capable of extracting salient features for place recognition. 2) A deep model trained from scratch and responsible for short-range visual navigation.

The third application demonstrates multiple robots all equipped with a camera, collaboratively navigating in an unknown environment. This method is inspired by ant-foraging behaviour and employs deep models proposed in the second application for salient feature extraction and short-range navigation. The system is evaluated in a 3D simulation via a set of experiments that demonstrate the effectiveness of the proposed method for collaborative visual navigation.

Keywords: robotics; navigation; deep neural networks; computer vision

To Bita, Maman, Baba, Faize and Farid

Acknowledgements

I would like to express my sincere gratitude to my senior supervisor Prof. Richard Vaughan for his support, guidance and immense knowledge. I thank you for listening and being open to all of my random ideas, though the research flow of the lab was different.

I thank my beloved Bita, I am so grateful for having you in my life. I thank you for your endless love, support and encouragement. I would like to thank my parents and siblings for their unconditional love, wise counsel and sympathetic ear. Words cannot express what I owe to you. You are the ones I have known and loved since day one.

It was a great pleasure for me to be a member of Autonomy Lab. I thank Pratik, Rakesh, and Payam for their suggestions and help. Amirmasoud for his fun and effective collaboration in semantic VTR. Rakesh, Jack, Pratik for proof-reading my thesis and giving me invaluable feedback. I would also like to thank all of my friends in the Autonomy Lab: Jack, Sepehr, Jacob, Lingkang, Geoff, Pratik, Payam, Rakesh, Bita and Amir for making this journey enjoyable.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Robots are widely used across society and industry, with many applications in manufacturing, transportation, service and cleaning. Navigation is one of the essential components of a mobile robot, making it an important topic for research. Navigation is usually paired with localization in a robotic system. As typically a robot must find its location in the environment to be able to navigate. There is no comprehensive solution to this task and navigation is categorized based on a variety of factors including the available sensors a robot is equipped with, environment, prior knowledge and so on.

Available sensors are either fixed-reference, such as GPS, or rate-measuring (accelerometers, odometry) that have no reference and should be integrated over time to obtain position. Both types have their limitations, like GPS being limited to certain environments where satellite waves are reachable, and rate-measuring sensors suffering from cumulative drift and divergence in localization-space. By contrast, images suffer neither constraint; being reference-free, and at the same time not suffering from drift making them suitable for navigation without requiring calibration (except for camera intrinsic and extrinsic). In some settings images can be individually informative enough to obtain position from. Although other localization and navigation solutions have used such as integration of visual odometry, an image is relatively unique to the location, camera pose and field of view that it was taken.

Various studies in psychology also suggest that animals rely on visual landmarks while navigating rather than a metric representation ([25, 29, 97]). According to a study [25]: *"Humans do not integrate experience on specific routes into a metric cognitive map for navigation. Rather, they primarily depend on a landmark-based navigation strategy, which can be supported by qualitative topological knowledge of the environment."*

In this thesis, we focus on *visual teach and repeat* for autonomous agents, which has various applications including surveillance patrols, inspection of structures and transporting goods. In this task, only visual data is available, and no other sensory information is provided. This limitation makes this task suitable for environments where Global Positioning System (GPS) data is too noisy or not available. One of the applications of this task

is to navigate the robot autonomously on the surface of Mars where other sensory data is either not available or challenging to use due to the different physical and environmental condition of the planet.

Visual teach and repeat has a teaching phase where the robot records visual information while being manually piloted over a path. The repeating phase is when the robot is asked to autonomously repeat the path given current visual information and the prior observations captured during the teaching phase. Classical approaches for feature extraction from images are often slow and not robust to various changes in the environment; such as illumination, viewpoint, lighting and other conditions which change pixel values a lot even when taken from the same scene.

Recently, Deep Learning has been widely used in the computer vision and robotics communities due to the exceptional success it achieved in a broad range of tasks such as object detection, face recognition, and image classification. This success is mainly thanks to the ability of deep networks (particularly convolutional neural networks) to automatically extract high-level features from a set of data, surpassing classical methods with hand-crafted features. It has also been shown [84, 10] that deep models are capable of extracting similar feature vectors of the same scene in different conditions and viewpoints.

In Chapter 2 we provide a demonstration of semantic features as condition- and viewpoint-invariant features for visual teach and repeat. An object detector model previously trained on an object detection dataset [67] is used in this work. We show that semantic features are robust and sufficient for monocular visual teach and repeat in our setting. We tested the method with real unmanned aerial vehicles in an office-like environment under different conditions. The contributions of this work are: 1) The first demonstration of semantic features for visual teach and repeat; 2) a novel motion controller that looks ahead in time to respond to upcoming objects; 3) sequential-temporal constraints for relocalization that stabilize the estimated position and lower the chance of perceptual-aliasing. The results of this work are published at the 15th Conference on Computer and Robot Vision [92].

In Chapter 3, we extend the work from semantic features to deep convolutional features and demonstrate a use case of deep features for both relocalization and motion control. Semantic features are shown to be robust and repeatable. However, system performance relies on the occurrence and variety of these features in an environment. Deep models are capable of extracting salient features along with low-level features and textures, so employing them in this task enables us to benefit from high-level and low-level features at the same time. A pre-trained place recognition network [9] is fine-tuned on a dataset collected in a photo-realistic simulator [78]. A visual servoing network is also trained from scratch on the same dataset. We evaluated our method in a simulated city-like neighbourhood on a long path.

In Chapter 4, the idea of using deep convolutional features is expanded to multi-agent navigation. Inspired by ant-foraging/trail-following algorithms, LOST (Localization-space

trails for robot teams) [96] and deep visual teach and repeat are joined to provide a new localization-space for robots to cooperatively navigate in an unknown complex environment. We evaluated the method with simulated agents in VizDoom [36] equipped with an RGB-D camera (depth images being used only for obstacle avoidance).

Chapter 2

Semantic Visual Teach and Repeat

In this chapter the use of semantic object detections as robust features for Visual Teach and Repeat (VTR) is presented. This work is motivated by the application of autonomous navigation in an GPS-denied environment where only visual information of a path is presented to the robot, and the robot should reliably repeat the same path. This work was the published 15th Conference on Computer and Robot Vision [92], and was implemented jointly by Ph.D. student Amirmasoud Ghasemi Toudeshki and Faraz Shamshirdar through pair programming under the supervision of Professor Richard Vaughan. One of the main challenges in this task is to design a system in a way that is robust to normal environmental changes such as lighting, appearance changes and camera viewpoint changes. Visual Teach and Repeat (VTR) has become a canonical task in robotics, and has many practical applications including surveillance patrols, inspection of structures and transporting goods. VTR has been studied using land, air and water vehicles. Here we consider monocular VTR on Unmanned Air Vehicles (UAVs).

VTR has two phases. In the teaching phase, a map-like memory is recorded from observations while the robot (or other training device) is piloted over the desired path. Then in the repeat phase the robot must (re)localize itself on the prior path, and repeat the remainder of it based on the prior observations in memory.

To investigate this problem we can break each phase into parts. Teaching phase is basically to encode visual information and store them in memory. These information later will be used by localization and the motion planner. Repeating phase can be broken down into two parts: *relocalization* and *motion planning*. A robot cannot effectively repeat the thought path without having a reliable localization. Once it can relocalize itself on the path, it can plan the motion accordingly. A reliable motion planning method is vital to the system too. Since taking a wrong action can take the robot far out of the path and this may lead to localization failure and the robot to be lost.

As discussed above all parts of a VTR system is important to be carefully designed. Visual information encoder in the teaching phase and relocalization in repeating phase are paired with an extra requirement of the visual encoder to be informative for motion planner

too. The two main categories for relocalization are (i) to make a 3D map using SLAM, with landmarks in 3D space; or (ii) to record a sequence of keyframes, with landmarks in image space. Approaches for the motion planner are vary and our choice on motion planner depends on the localization method and the type of information stored in memory.

Our approach to problem of encoding visual information into memory is to use only sparse semantic object features. Recent work on Convolutional Neural Networks showed significant success on training models that can detect multiple objects of tens or hundreds of categories in an image at frame rates. We show that such detections are repeatable enough to use as landmarks for VTR, without any low-level image features. Since object detections are highly invariant to lighting and surface appearance changes, our VTR can cope with global lighting changes and local movements of the landmark objects as well as the camera viewpoint changes.

With such a powerful and robust visual encoder we could have the same set of information in different lighting situation, different seasons and environments with dynamic objects within where most of low-level feature detectors fail to detect the same set of features. Regarding the selection of semantic object detector as our visual encoder, we used overlap of the objects" bounding boxes with the same class label as a measure to estimate the similarity of two keyframes. Spatial-temporal constraints inspired by Sequential SLAM [47] are also applied to enhance the accuracy and performance of relocalization. Using a temporal sequence of landmarks reduces the effect of perceptual aliasing caused by sparse landmarks and a limited landmark vocabulary.

Our choices of motion planner is limited because of the chosen visual encoder method. During the repeating phase motion planner will acquire current observations of the environment as well as the desire keyframe provided by relocalization. These observations are bounding boxes of detected objects with their class labels in both keyframes. By the two, motion planner should take an action that will pass the robot through the thought path. Even though the visual information are powerful enough to manage camera viewpoint changes and different environmental conditions, but taking a bad action may fail the system entirely. Our motion controller is based on the *funnel control* theory [11] with a novel extension that exploits the scene memory to anticipate the future.

Later in this chapter we will discuss work related to semantic visual teach and repeat in 2.1. Our method and system design are presented in 2.2 followed by experiments in 2.3. We will finally finish this chapter by a conclusion in 2.4 and discussion and possible future works in 2.5.

2.1 Related Work

Barfoot has a significant body of work on Teach and Repeat navigation, including [12] which tested a ground robot in a path over 1km, demonstrating a color-constant method to handle

changes in illumination. However, this work is sensitive to changes of viewpoint and hence requires the initial position to be similar. To handle gradually-accumulating environmental changes, [45] employs a “multi-experience" method for VTR, which was introduced in [63].

Visual teach and repeat using UAVs is only recently feasible, with few early examples [66] [55]. In [66] the authors demonstrated a quad-rotor with downward-facing camera that repeats a simple path starting from the same position with no changes in the environment. The method is similar to [28] and based on SURF descriptors and pose-based localization. Nguyen et al in [55] also choose SURF features for descriptors but their localization technique is appearance-based. Experiments are in simulation, using an AR-Drone model in Gazebo, and again the environment is unchanged between teach and repeat.

Surber et al in [86] build a map using structure from motion (SfM) from data captured in the teaching phase and execute visual-inertial odometry on a real UAV in the repeating phase for localization, using a previously constructed map as prior knowledge. They use BRIEF descriptors and report that relocalization is successful only where the appearance is similar enough to the reference map.

Krajnik et al in [42] examines combinations of feature detectors and descriptors for VTR, favouring the STAR feature detector and GRIEF feature descriptor for good accuracy with computational speed. GRIEF is a binary feature descriptor intended to be robust to seasonal changes in appearance.

So-called “semantic" segmentation of foreground objects from images is a traditional topic in computer vision and robotics [69], [74]. The recent development in deep neural networks, particularly CNNs has this area flourishing [84, 53, 89, 5]. New object detection algorithms provide much better accuracy and speed than was previously available [67, 68, 43] which inspired our attempt to use them as online real-time feature detectors

2.2 Method

2.2.1 System Overview

Figure 2.1 outlines the structure of our system. During teaching, the UAV is flown or carried along the target trajectory. VTR system runs on a stationary computer equipped with GPU along with YOLO-2 CNN object detector. it receives video frames from the UAV periodically, passing them to the detector. The CNN output is processed to obtain a scene descriptor of the locations of objects and their class labels in the image. Each descriptor is stored in sequence. The finished sequence is the robot’s memory of the learned trajectory. The rate of storage set to be real-time to make sure all visual data are being stored in memory. However, for memory efficiency and processing time of the system, a non-real-time rate can be set. Trade-off between choosing a small and large frame rate is to have all important visual sensory information or have a smaller memory and faster relocalization.

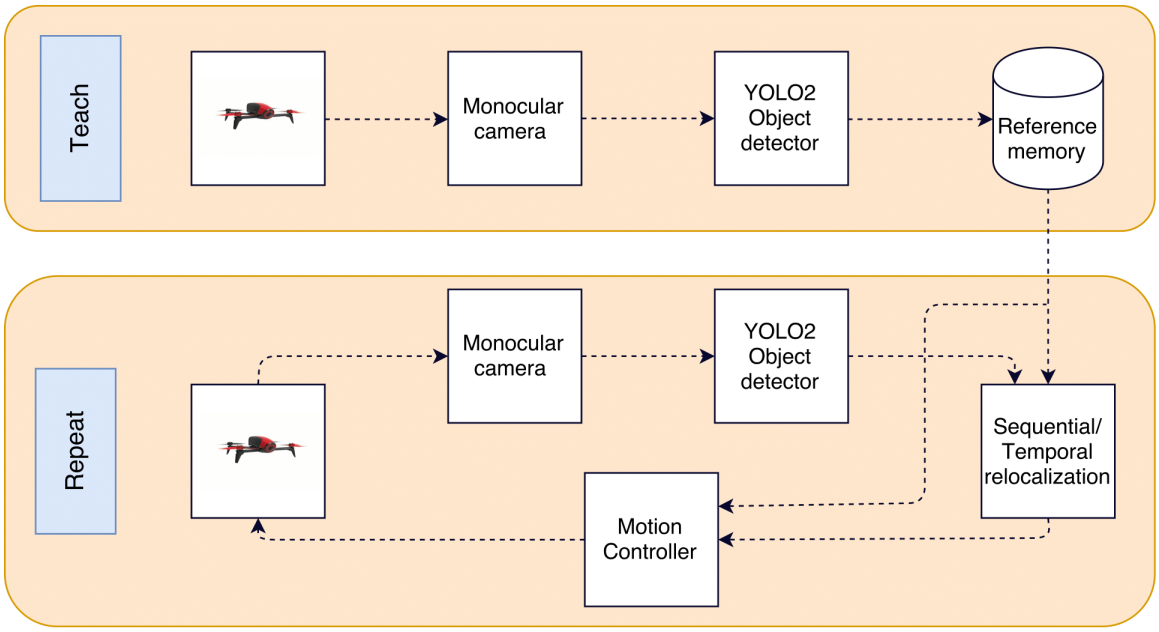


Figure 2.1: System Overview [92] (Copyright ©2018, IEEE)

In the repeat phase, the UAV motion is actively controlled by the VTR system. Again, images from the robot input to the object detector to obtain a scene descriptor. The re-localization module then finds the closest match between the current scene descriptor and memorized ones. Once localized, motion controller attempts to move the robot so that the next scene will look like the next scene in the stored sequence. This action will be sent back to the robot to be taken. Communication between the robot and stationary computer is over WiFi. These modules are described in more detail below.

2.2.2 Detection

YOLO-2 [67] CNN object detector is being used for object detection. This model is trained on the COCO dataset. It is notable for its speed in comparison with other deep networks like Faster RCNN [68]. We can detect objects at 40fps on a commodity PC with NVIDA-GTX 1080Ti GPU.

YOLO-2 has different pre-trained models that are vary on the number of object categories, accuracy, size of models and processing time subsequently. It can detect a variety of common objects and animals, Figure 2.2 shows an example of detections on an image made by YOLO. YOLO was not designed for UAV landmark detection. We use YOLO and an office/lab environment that contains objects detected by YOLO as a placeholder for a dedicated UAV landmark detector and outdoor flights in this initial study. The system should work as described without modification in real-world application environments given an appropriately trained feature detector. For example, VTR across a city park might use

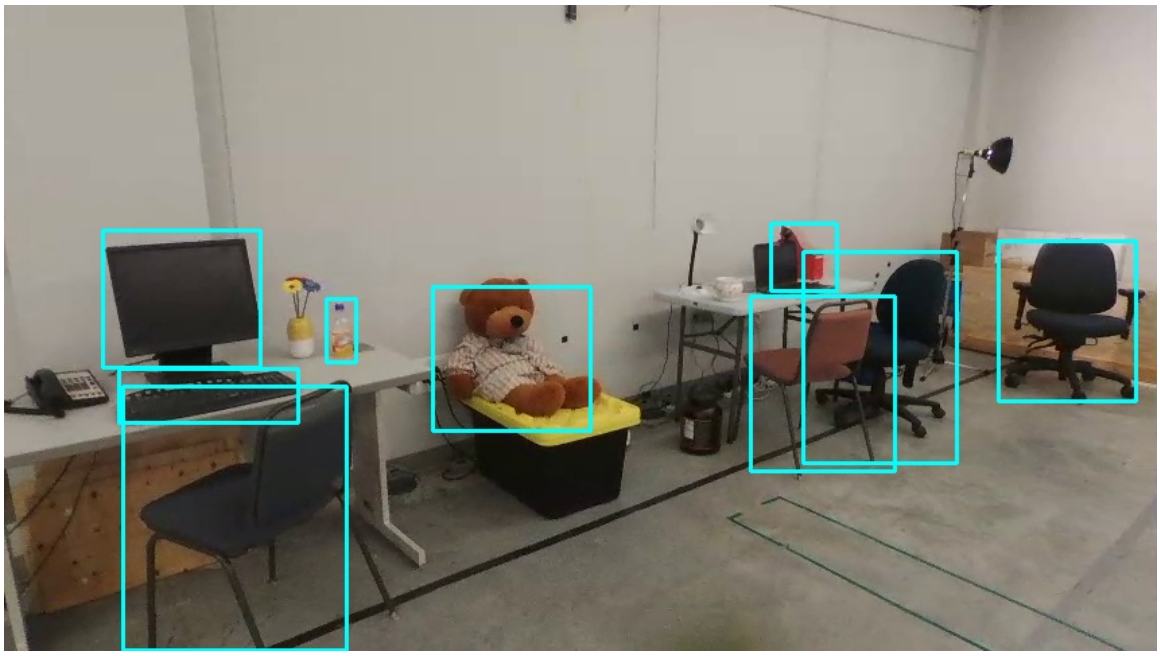


Figure 2.2: YOLO-2 predictions on an image with object labels predicted correctly.

signposts, path junctions, benches, fence posts, flower beds, tennis courts, and swimming pools as semantic features.

2.2.3 Reference Memory

Object detector model provides a bounding box, confidence and class label for each detected object. Detections below a confidence threshold are discard (we used confidence ≥ 0.55 in our experiments). Some object labels that are not reliable landmarks such as *person* and *cat* are discard too since their world positions are not fixed. Each detection is stored as a vector of five floating point values describing the object category and bounding box in the image. We found that each scene might contain two to eight objects, so the resulting whole-scene descriptor is small: of the order of $5 * 8 * 4 = 160$ bytes. For comparison an single original SIFT point descriptor is 512 bytes (both using 32-bit floats), and in typical use scenes contain 500 or more points, for 256,000 bytes per scene. Our scene memory is thus around 1600 times smaller than a SIFT-based method while achieving reliable VTR.

2.2.4 Relocalization

So far we discussed how a reference memory will be created upon recording visual data during the teaching phase. Based on the data stored in reference memory, robot should relocalize itself within the sequence of descriptors in reference memory. However, Robot's initial location may be far from the original taught starting location. Also, in a real world application, environment may change over time. For example some objects may get removed

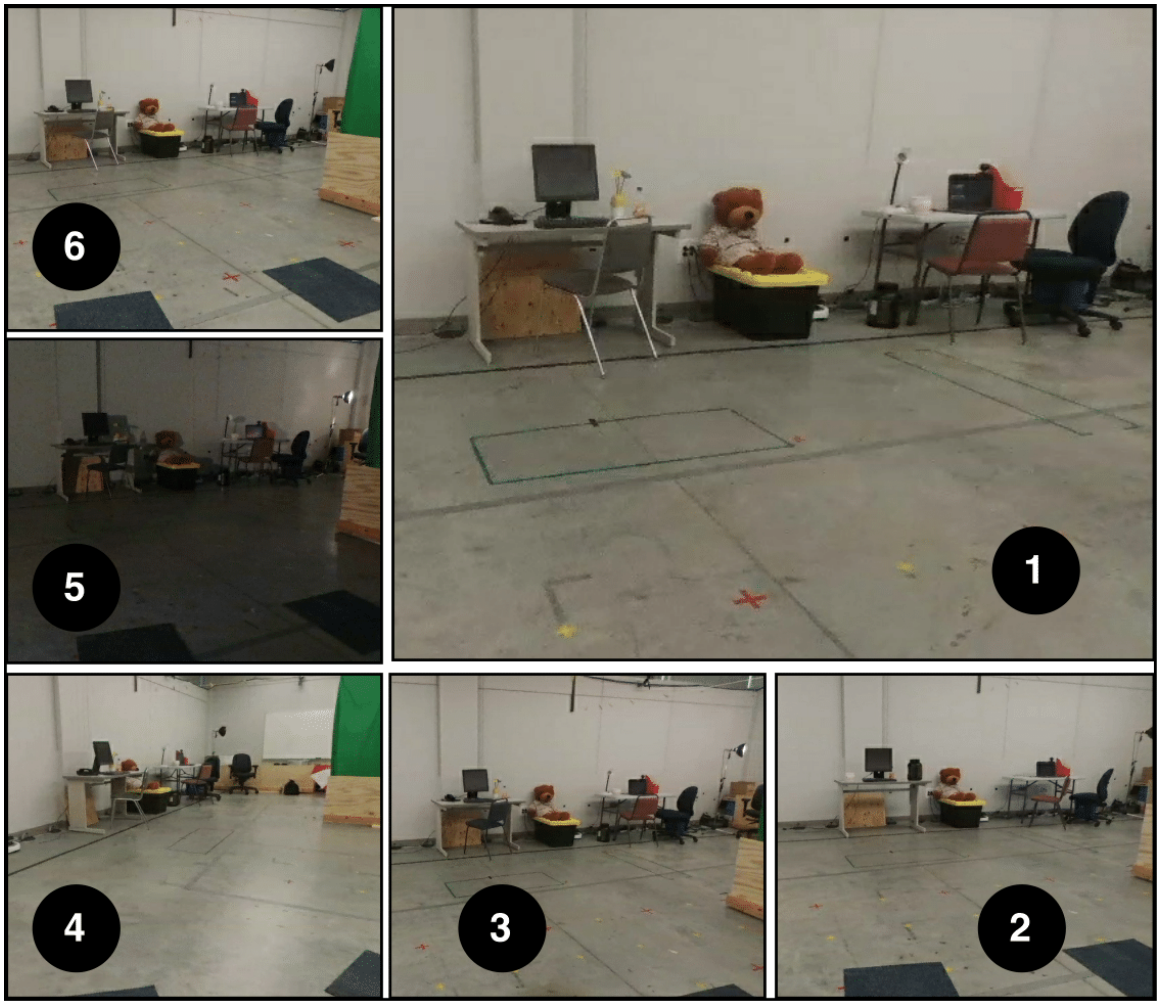


Figure 2.3: Relocalization robustness: (1) is the first image in the taught sequence; (2-6) are examples where the robot has successfully relocalized despite large changes in viewpoint and scene contents and appearance by matching a sequence of object locations (chair, screen, bear, etc.) from a sequence of descriptors stored during teaching and correctly located (1) as the closest matching image. Note the missing and moved objects in the repeats. [92] (Copyright ©2018, IEEE)

or moved around, lighting may be different during a day. In all of these cases, robot should still reliably find the correct match in order to take a proper action.

Similarity measurement used in our approach is first to find the similarity for two individual object detections by calculating area of overlap of their bounding boxes to the sum of their bounding box area:

$$S_O = \frac{(\alpha s' \cap \alpha s'')}{\alpha(s' + s'')} \tag{2.1}$$

Where s' and s" denotes the area of the bounding boxes of objects recorded in the target and current images respectively. α is a weighting factor to tune matching sensitivity. Two

individuals may not have an overlap between their bounding boxes because of many factors including environment changes, taking wrong actions and leaving the path or starting from a different location. Having an α greater than 1 in equation 2.1 results in increasing the size of bounding boxes so that an overlap may occurred. We chose this by experiment, to get a working value of 4.

Object association is a big issue in the case of having a descriptor that has two or more objects with same class label. Distance of objects in two frames and their size is a good factor to find the association. Since the similarity measurement conveys these factors implicitly, object association in our approach relies on the similarity of two objects. We first calculate all the similarity of objects with same class label and then ones with higher similarity will be associated. Overall similarity of keyframes is then calculated by averaging over all the similarities (S_O):

$$S_I = \frac{\sum_{n=1}^{N} {S_O}^n}{N}, \tag{2.2}$$

where N is the number of objects in the scene.

Compared to point-feature approaches, we have very few features in each scene, and a dictionary of only a 80 object categories. Thus scene descriptors are not highly discriminative: in our setting lots of scenes contain a chair below a keyboard below a screen. This problem is also present in methods where low-resolution whole-images are used as descriptors, so we borrow the SeqSLAM technique developed for that domain [47] where localization is done over a contiguous sequence of images. The last N observations are compared with the memorized observations and a score table is recorded. Since the robot may have different velocities in the teach and repeat phases, we must test different velocities to find a good match. Figure 2.4 shows an example score table and line segments corresponding to different UAV velocities. We find the line that passes though score cells with the highest sum. The gradient of the line gives the UAV velocity and the peak value along the line gives the current position along the trajectory.

Although sequential matching helps the system to reduce the effect of perceptual aliasing, we can also benefit from the fact that matching has to be continuous in time. Since the taught path is recorded over a continuous period of time, we know that robot can not jump over in time. To address this, after initial relocalization, we positively weigh descriptors nearby in the sequence to add a temporality constraint. The more confident the previous match, the narrower is the distribution we weigh nearby descriptors with. Also, while testing relocalization, we found that in trajectories with repeated sections (time-extended loop-closures), basic sequence matching could match the wrong location and the robot could become stuck in a loop. With this mechanism, the robot can correctly repeat trajectories with repeated parts.

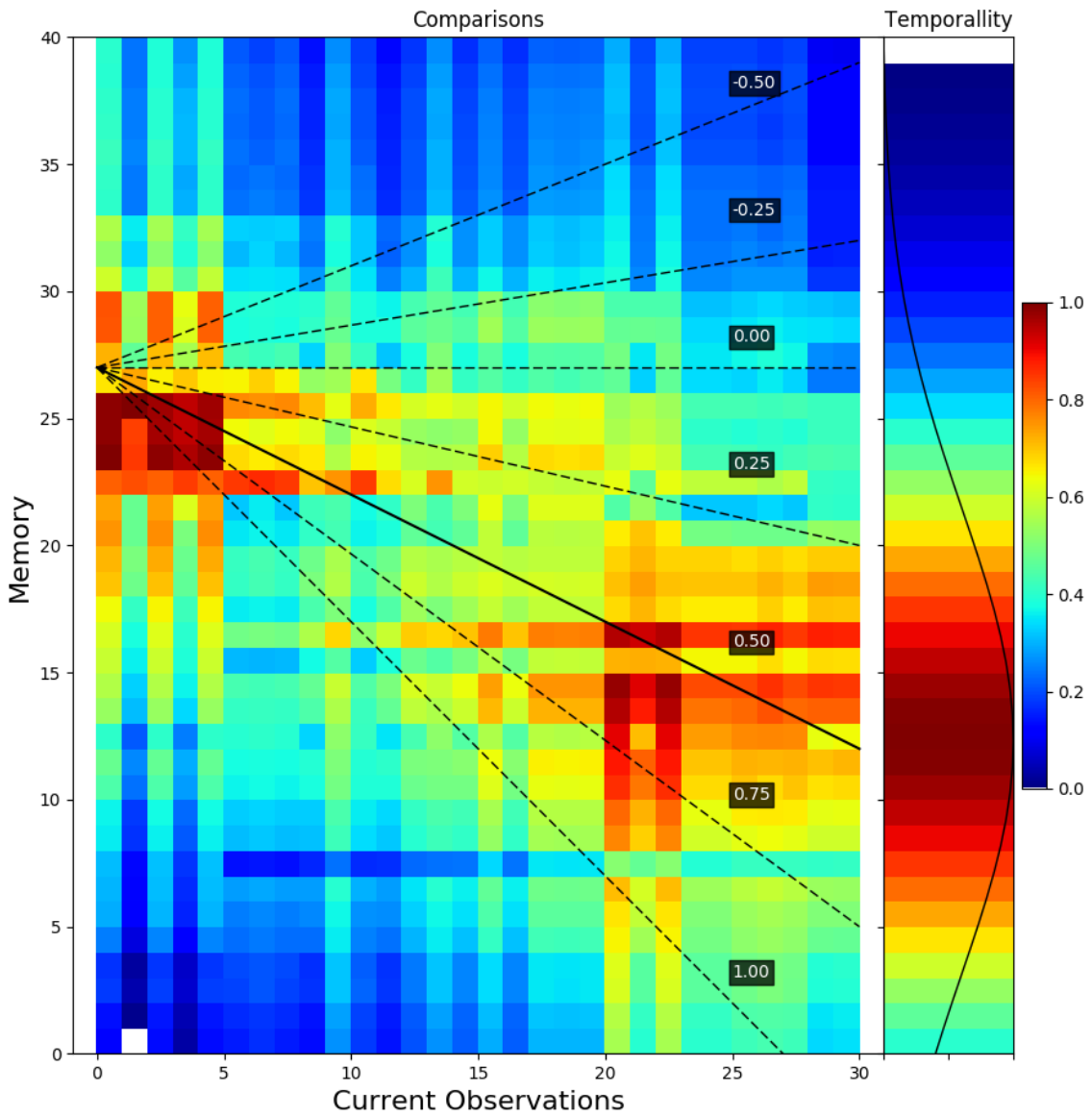


Figure 2.4: Extended SeqSLAM relocalization method: heatmap of the similarity of a sequence of 10 recent scene descriptors compared to memorized descriptors in reference memory. The line segment with the highest sum of values gives the best estimate of current UAV position and velocity. After initial relocalization we weight nearby scenes to impose a temporality constraint that avoids being confused by time-extended loop-closures. [92] (Copyright ©2018, IEEE)

Adding the temporality constraint, the similarity measure becomes:

$$S_t = \frac{S_s(1 + \beta \cdot \text{norm}(x = i, \mu = i', \gamma))}{(1 + \beta)}, \tag{2.3}$$

Where S_s is the score of matching that has been done by sequential comparison and β is the factor of temporality. β is selected by the confidence of previous match. i is the index of the memory which is being calculated and i' is the average of the last 5 matched indices of the memory.

2.2.5 Motion Control

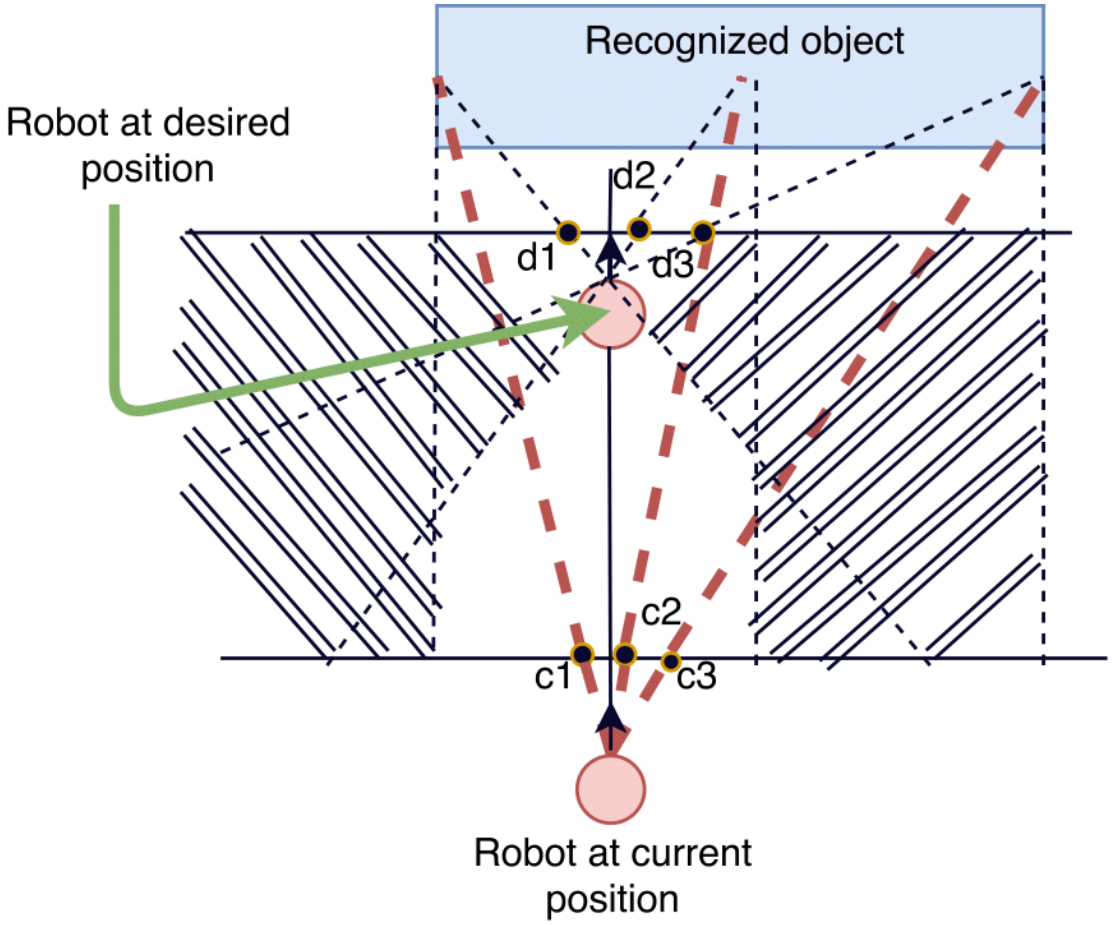


Figure 2.5: Schematic Funnel Controller adapted from [11]: the controller predicts action given the current robot position, desire position, and recognized objects. [92] (Copyright ©2018, IEEE)

Our learned trajectory is in the form of object locations within keyframes, so we use a following controller afforded by this information. “Funnel lane" path following controller [11] method uses image-space point landmarks to achieve robust visual servoing. We extended it here for the case of object landmarks. We control robot pose on a 2D plane of constant altitude only, but the method is trivially extended to altitude control. Three points are considered for each object detection: one at the center and one at each extreme of its bounding box in the horizontal axis. Using the extent of the object allows us to reason

about its scale change (but not its size in 3D). Figure 2.5 sketches the geometry of how two detections of the same object can be used to construct a "funnel lane' that guides the motion of the robot to obtain the second camera position given the first.

One object detection is enough to repeat an (x, y) trajectory with respect to that object, but the approach direction is not constrained. A second object allows us to control yaw, to get replay of the full sequence of 2D poses (x, y, θ).

Equation 2.4 shows the funnel lane controller response for the single landmark on the left x-extent of the recognized object in Figure 2.5. c_1 and d_1 are the horizontal distance of the land mark in image space of the current position and desired position respectively, and $\phi(c_1, d_1) = \frac{1}{\sqrt{2}}(c_1 - d_1)$.

Equation 2.5 averages the response for a whole object over its three landmarks. The final response is a command for heading adjustment. We scale the output by a constant gain for tuning (we used a factor of 12).

$$\theta_{c_1} = \begin{cases} \gamma \min\{c_1, \phi(c_1, d_1)\}, & \text{if } c_1 > 0 \text{ and } c_1 > d_1, \\ \gamma \max\{c_1, \phi(c_1, d_1)\}, & \text{if } c_1 < 0 \text{ and } c_1 < d_1, \\ 0, & \text{otherwise.} \end{cases} \tag{2.4}$$

$$\theta_{\text{obj}} = \frac{(\theta_{c_1} + \theta_{c_2} + \theta_{c_3})}{3}. \tag{2.5}$$

Often, several objects are detected in a single scene. When this occurs, the controller calculates the response for each object individually using Equation 2.1. Then it performs the funnel lane method for all objects as sketched in 2.5. We can then compute the response for the entire image by averaging objects responses:

$$\theta_{\text{image}} = \frac{\sum_{n=1}^{N} \theta_{obj}}{N}. \tag{2.6}$$

We then perform Equation 2.6 on set of frame starting from the best matched to N future frames to align the robot with the best matched frame as well as looking ahead to align with upcoming frames:

$$\theta_{\text{Funnel}} = \frac{\sum_{i=I_m}^{W_f} \theta_{\text{image}}(\text{current}, i)}{W_f}. \tag{2.7}$$

Where I_m is the index of the matched keyframe and W_f denotes the size of funnel control window. We used $W_f = 30$.

The basic funnel controller has a failure mode that is difficult for our application: it steers the robot to align the currently-perceived objects as they should appear in later frames (in

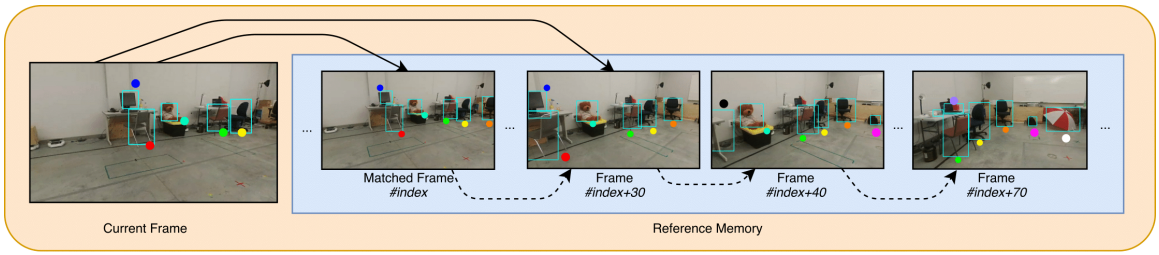


Figure 2.6: Funnel Lane with short look-ahead window plus Virtual Funnel Lane with long look-ahead window to react to unseen but predicted objects. [92] (Copyright ©2018, IEEE)

short look-ahead). But it does not react to upcoming objects that can not yet be seen. Thus with a narrow field of view camera it fails to turn the robot in sharp corners. To address this, we augment the funnel lane controller with what we will call a *virtual funnel lane*: we simulate the action of the normal funnel lane controller by running it on the sequence of stored images from the current best match up to some window look-ahead size. The robot simulates what will happen in the future, assuming it is correctly localized. This was the robot can “see around" upcoming corners. The robot records the heading changes prescribed by the controller in this virtual look-ahead flight, and averages them to obtain a predicted future yaw value as described in Equation 2.8. This is then blended with the original funnel lane control by Equation 2.9. The intuitive effect of these two control components is that the “real" funnel lane aligns the robot well to the things it can see now, while the virtual funnel lane pre-aligns the robot to objects it can not yet see. Thus the robot will turn nicely around corners even when the set of objects in view changes completely.

$$\theta_{\text{Virtual}} = \frac{\sum_{i=I_m}^{W_v} \theta_{\text{image}}(i, i+1)}{W_v} \tag{2.8}$$

Where I_m is the index of the matched keyframe like equation 2.7 and W_v denotes the size of the virtual funnel window. We used $W_v = 70$.

$$\theta_{\text{total}} = \alpha \cdot \theta_{\text{Funnel}} + (1 - \alpha) \cdot \theta_{\text{Virtual}} \tag{2.9}$$

Where θ_{Funnel} is the weight of funnel controller range between 0.0 to 1.0, that indicates the importance of funnel controller and virtual funnel controller respectively. We chose this by experiment, to get a working value of 0.7.

This enhanced controller is illustrated in Figure 2.6. The current image is only being compared with the next 30 key-frames in the memory to generate a funnel lane control command, but the virtual funnel lane looks ahead 70 key-frames to consider objects not yet in view such as the umbrella. The look-ahead anticipates the turn towards the umbrella and forces the controller to change the robot's heading towards it.

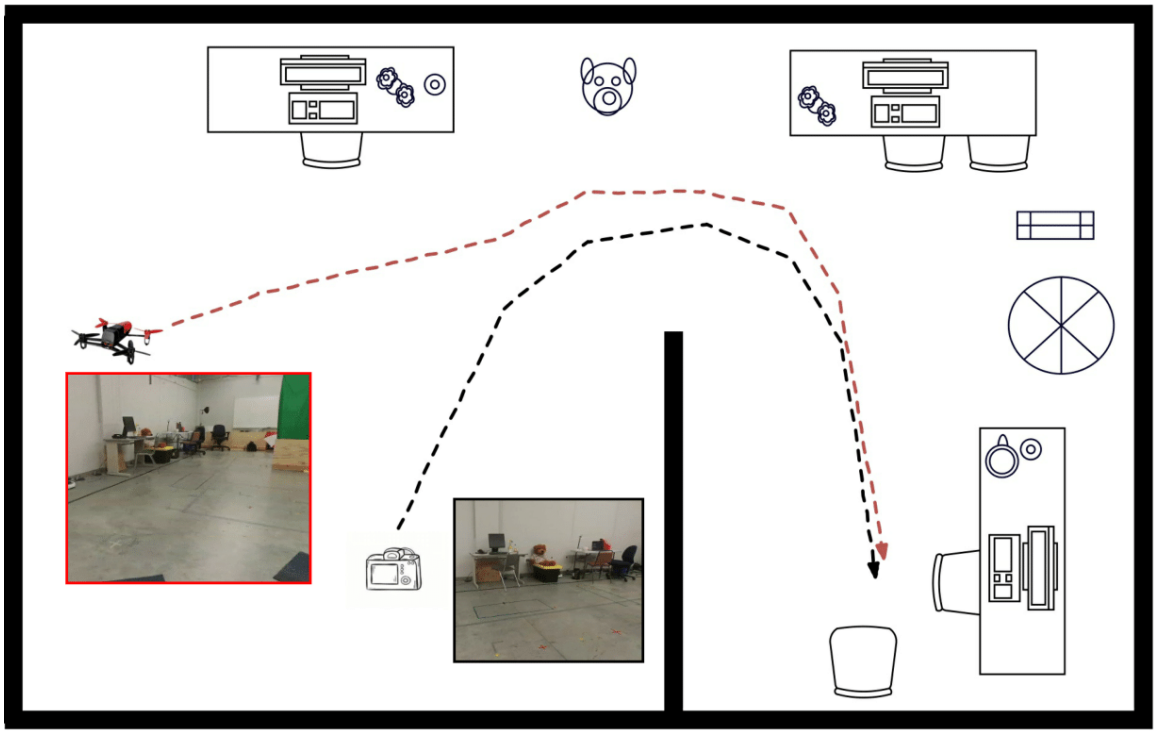


Figure 2.7: Test scenario: The trajectory of the camera is encoded as a series of object landmark locations in image space. Later, the UAV autonomously localizes itself in this space, then repeats the remaining part of the camera trajectory. A CNN detects objects such as desk, chair, mug, umbrella, backpack in keyframes. The UAV trajectory is generated directly from pairs of keyframes. [92] (Copyright ©2018, IEEE)

To complete, control logic is: (i) rotate on the spot if not well localized; (ii) if yaw error is below a threshold, go forward with constant velocity, else yaw to correct the error. We decoupled yaw and forward motion as the low-level flight controller behaved badly when attempting to yaw and translate at the same time due to the complex vehicle and controller dynamics of a quad-rotor.

2.3 Experiments

We performed real-world experiments with a commodity Bebop-2 UAV. We are limited to a 10m x 6m experimental arena where we can independently measure UAV trajectories using a Vicon motion-capture system. The test environment is pictured in schematic in Figure 2.7. We limited the total number of recognizable objects in an office-like environment to 18 including chairs, tables, monitors, and so on.

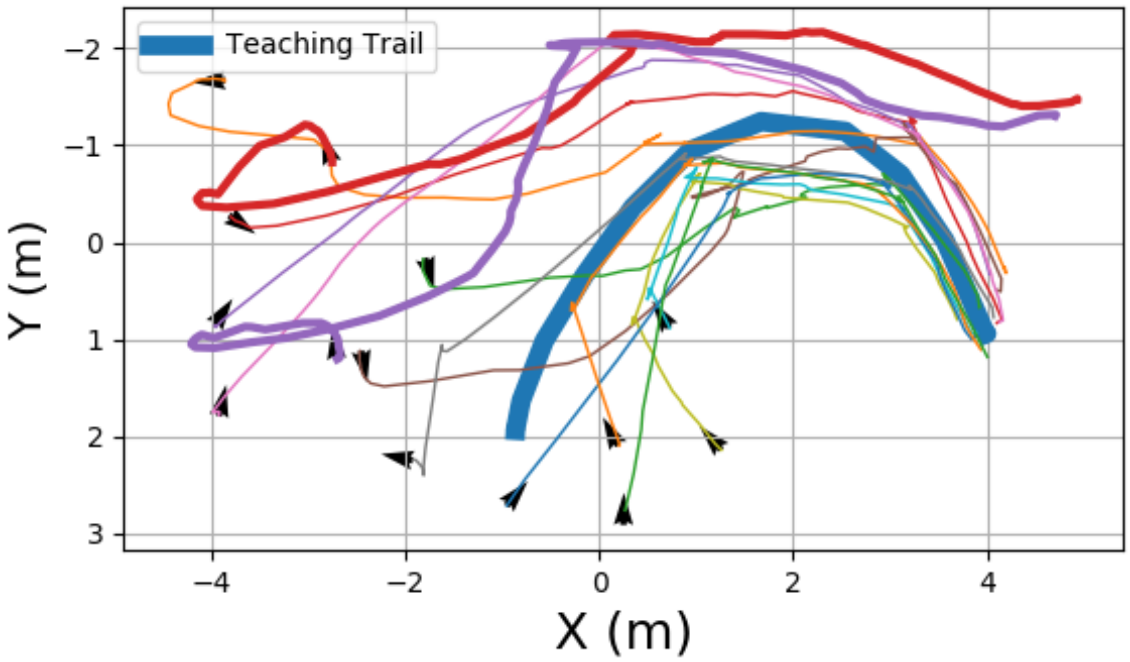


Figure 2.8: Trails in both teaching and repeating phases during experiment I, arrows are indicating initial rotation of the robot. [92] (Copyright ©2018, IEEE)

2.3.1 Teaching

In the training phase, the UAV is used as a passive camera. It is placed on a wheeled cart and pushed along the desired route, indicated by the trajectory marked with a camera Figure 2.7. The altitude is constant at 1.5m. The pose is recorded independently by motion capture for further analysis. The video is fed to the CNN object detector and descriptor generator and the sequence of memory of 290 key-frames is built.

2.3.2 Experiment I: Changing Viewpoint

A first experiment was performed to test the robustness of relocalization and trajectory repeating to changes in initial camera viewpoint. We run 12 trials with the the robot starting from (x, y, θ) poses chosen at random up to 5m away from the original taught location, and one of two fixed altitudes chosen at random. The robot flies autonomously, controlled by the VTR system. On startup, the robot rotates on spot until relocalization is achieved, then attempts to repeat the trajectory until matching the last keyframe.

Results

In 10 out of 12 trials, robot has completed the learned trajectory, arriving less than 0.5m from the original end-point. Figure 2.8 reports the start positions and the paths of all robots. The blue bold line is the taught path. Note that the robot navigates around an opaque wall

(Fig 2.7), so that the features detected in the second half of the trajectory are not visible in the first half: direct visual servo to the final target can not solve this task. In both the failed cases, the robots incorrectly detected the endpoint keyframe too early, stopping at the pair of chairs at the top right, and not the correct final pair of chairs at the bottom right. Although, sequentially and temporality constraints are added to overcome these perceptual aliasing problems, but in this case descriptors were similar in sequence and close in time. Both trails did correctly localize initially and completed 65% of the trial correctly.

2.3.3 Experiment II: Changing Environment

A second experiment investigates the ability to repeat a learned trajectory when the environment changes. The robot starts approximately at the same place as in teaching. But in one set of trials we change the lighting by turning off the ceiling lights and turn on two spot lamps. In a second set of trials we remove five objects that were noted by the object detector in training, and move eight others. The supplementary video shows us disturbing the objects to change their appearance. We repeat the trajectory and record the time taken to arrive at the final landmark.

Results

Examining robustness to appearance changes, we record the time taken to repeat the trajectory after (a) drastically changing the lighting and (b) removing objects and moving others to change their appearance. A histogram of flight times is shown in figure 2.10 with fitted Gaussians.

Removing and moving objects changes the execution time distribution, increasing it by an mean of 4sec (less than 10%).

But, perhaps surprisingly, changing the illumination reduces time taken for a repeat. The data show that the changed illumination slightly reduced the number of recognized objects, in particular the objects that were less repeatably detected in the teaching phase. Without objects disappearing and reappearing as frequently, the flight controller produced smoother behaviour and reached the goal faster. Perhaps the method could be improved by deliberately filtering such “flickering" objects.

Figure 2.9 provides evidence of this effect, showing that the number of detected objects differs during the light-changing and removing/moving objects experiments. When removing/moving objects, the number of detected features decreases, but the run time is longer because the moved objects degrade the controller behaviour, and the robot takes longer to line itself up along the trajectory. In the spot-lighting trials, the fewer object detections improve completion time as described above.

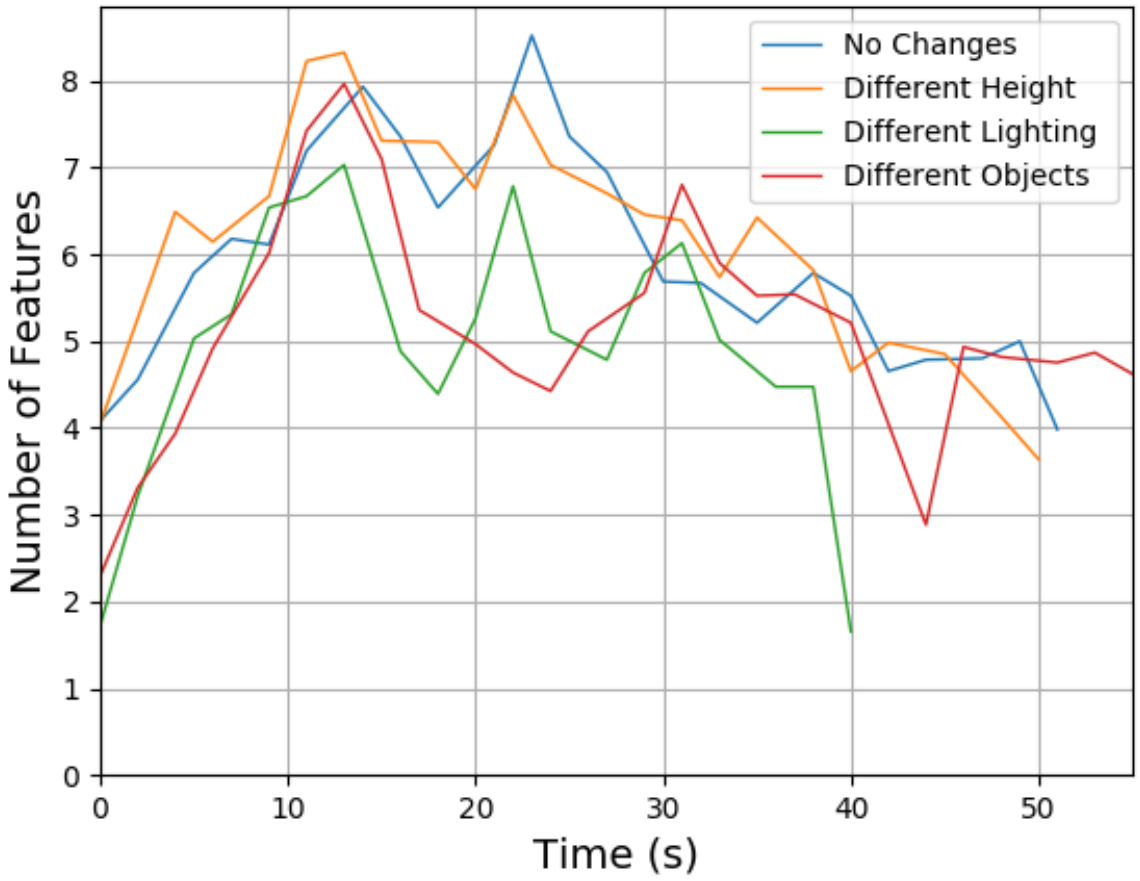


Figure 2.9: Number of features detected at nearest-corresponding locations during teaching and repeat phases. [92] (Copyright ©2018, IEEE)

2.4 Conclusion, limitations and future work

The work presented in this chapter showed that a current deep-CNN object detector can be used to provide robust and repeatable features that are sufficient for monocular VTR. We demonstrated the method with real UAV experiments, but in an lab/office-like environment as a substitute for our eventual goal of large scale outdoor flights. The indoor environment allowed us to use a pre-trained object detector since a detector for UAV navigation tasks is not available, and allowed us to collect ground truth data from an installed motion capture system. Objects are detected with viewpoint and lighting invariance that compares favourably with other methods, and with interesting novel invariances to the object being completely turned around or replaced by another of the same category. We demonstrated that we can start the robot 5m away from the taught initial position, facing at a random heading, and the robot could relocalize itself and complete the trajectory using the object detections alone. Comparable experiments using SURF features have been shown to work at not more than 1m disturbance. We also successfully repeated a trajectory with the UAV's

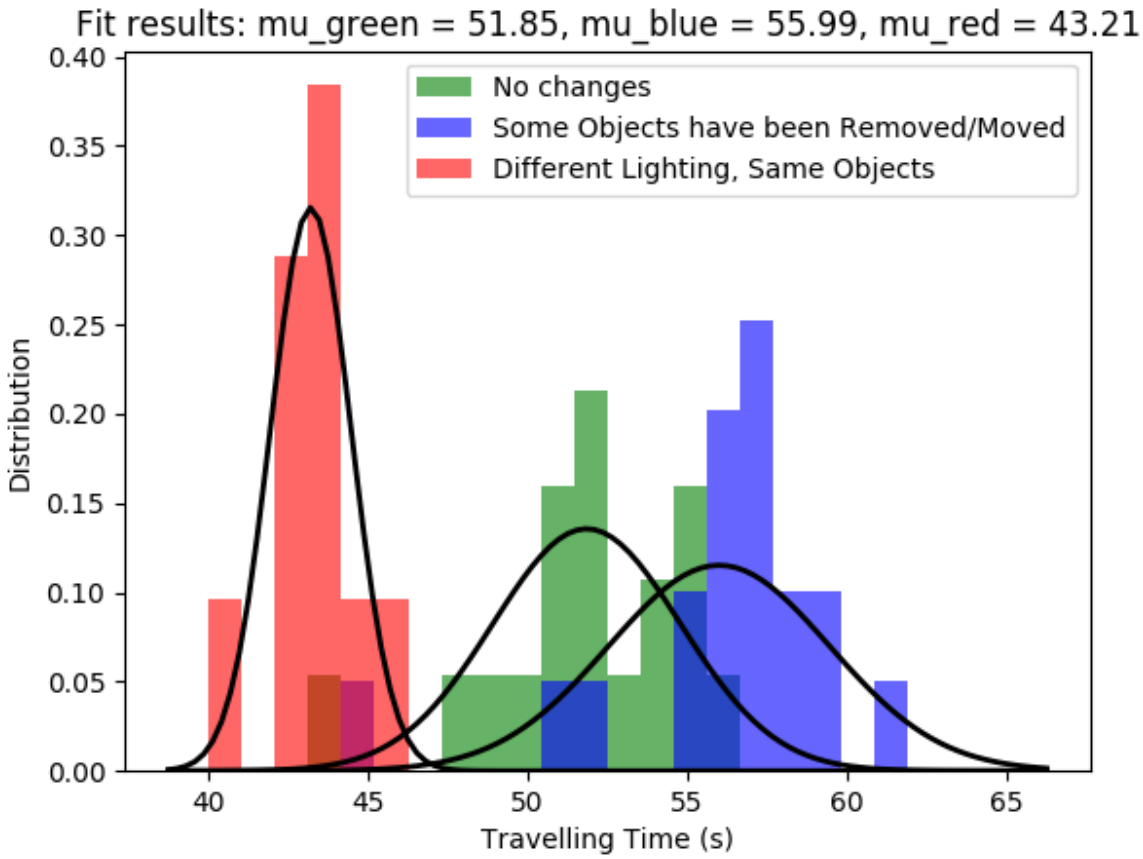


Figure 2.10: Repeat completion times for identical environment (green), light-changed environment (red) and object-moved environment (blue). [92] (Copyright ©2018, IEEE)

altitude changed by 40%, and also if several objects are removed and removed, or if the global lighting is changed dramatically.

We also contribute a novel two-window use of the funnel lane visual servo method that looks ahead in time to respond to upcoming objects.

Our method depends on a supply of distinct ambient objects. It will not work in environments without discontinuous, recognizable object categories. Previous work on automatic generation of low-level feature dictionaries could suggest directions for future work. For taking this work outdoors over kilometer scales, we need a CNN that reliably detects objects that exist in our target environment. We may have to train our own networks on the salient objects. In settings where distinct objects are rare, we could extend the method to use point features and/or inertial sensing to allow navigation in object-free areas.

One simple but interesting extension of the work would be for the robot to announce changes to the set of objects. If a previously-seen vase or painting is missing, perhaps it has been stolen. If a backpack has appeared, perhaps it was left accidentally. More substantially, perhaps a robot is performing an inspection trajectory over a new manufactured object such as an aeroplane wing. Is a rivet missing or has a bad weld appeared, compared to the teach

phase recorded over a known-good example? If patrolling a construction site at night, a newly arrived truck or absent machine could be a security issue.

The recent performance improvement in some computer vision tasks due to CNNs and similar methods are giving us interesting new choices for robot vision. We aim to exploit the new robust vision methods towards more complete robot autonomy and new applications.

2.4.1 Contribution

This project was implemented jointly by Ph.D. student Amirmasoud Ghasemi Toudeshki and Faraz Shamshirdar from Autonomy Lab, Simon Fraser University through pair programming under the supervision of Professor Richard Vaughan. Faraz's contribution was mainly on code implementation, detection, localization, and virtual look-ahead controller while Amirmasoud's contribution was on the general idea and the funnel-lane controller. Experiments, analysis and literature review were done equally by Amirmasoud and Faraz.

Chapter 3

DeepVTR: End-to-End Visual Teach and Repeat

The work presented in this chapter is an extension to the previous chapter where we employed only semantic objects as the mean of scene description for visual teach and repeat tasks. Although the method performed well at relocalization and motion control in a indoor environment, it is limited in environments without discontinuous recognizable object categories. Also, apart from relocalization, because of the 2-dimensional nature of semantic features in a scene, our motion controller was limited to only managing the heading of the robot (yaw).

End-to-end deep learning approaches inspired by navigation in animals have recently been proposed to learn navigation in complex environments. These approaches can be divided into three categories: 1) reactive and memory-less [20, 103], 2) based on general-purpose memory such as LSTM [50, 49], and 3) based on a navigation-specific memory [60, 32, 75]. Our previous work on semantic visual teach and repeat was not a learning approach, but it can be categorized as belonging to the third category since we employed object detections as visual landmarks.

Following the previous chapter, one of the main requirements of a VTR system is a relocalization and a motion controller that is robust to normal environmental changes such as lighting, appearance changes and camera viewpoint changes. The employed object detector was robust to these changes, but it has some disadvantages: (i) lack of detectable objects in different environments such as outdoors (ii) they are not highly descriptive and may cause relocalization to fail. These limitations make this approach less applicable. We need a visual encoder that can exploit salient features of an image and not only the detectable objects. It should also be condition- and viewpoint-invariant. Deep learning has been used as an approach to extract such features for different tasks such as camera relocalization [37], place recognition [9], face recognition [76] and representation learning [2]. Representations generated by a CNN are capable of encoding salient features of an image with respect to

an objective. For instance, not only can it encode the presence of a chair in a scene but can also encode salient features of the chair.

Our new approach to visual teach and repeat consists of three modules: 1) PlaceNet: a deep visual encoder network for salient-feature extraction 2) Relocalization 3) LocoNet: a deep motion controller network for short-range navigation. PlaceNet extracts salient features of a scene as an internal representations. It builds embeddings out of images recorded during the teaching phase and stores them in memory to be used by relocalization module later on while repeating the path. The relocalization module employs sequential-temporal constraints on embedding-based similarity measurement to relocalize the robot on the path. Once relocalized, a reachable target image is selected and fed into LocoNet along with the current image to plan an action.

We discuss work related to deep learning approaches in navigation in Section 1. Our method and system design are presented in Section 2 followed by experiments in Section 3. We will conclude this chapter in Section 4 and discussion and possible future work in Section 5.

3.1 Related Work

As we discussed in the previous chapter, Barfoot has a significant amount of work on the teach and repeat navigation, but his work such as [12, 28, 99] mostly relies on classical approaches on localization and place matching. There is also other work from the community on visual teach and repeat by classical methods such as [42] that employs STAR feature detector and GRIEF feature descriptor or [28, 55] that use SURF descriptors and pose based and appearance based localization.

Going beyond visual teach and repeat, there are an enormous number of studies on navigation in psychology. Tolman in [91] conducted experiments on navigation in rats and showed that they build an internal representation for space in the form of cognitive maps, enabling them to reason about the map and build shortcuts. Keefe et al in [58] suggested that internal representations are in the form of a metric map. However, [54, 87, 31] and [8, 23, 22] have shown that rats and honey bees mostly rely on landmarks while navigating. This idea is also suggested for human navigation. In [25], the authors found that "participants could not take successful shortcuts in a desert world but could do so with dispersed landmarks in a forest' suggesting that humans are not likely to build cognitive maps from path integration, but instead, rely on landmarks when they are available. The idea of landmark-based navigation in human is also supported in [29, 97].

In classical robotics, localization and mapping for navigation is a well-studied problem. The indoor-navigation systems proposed fall mainly into three groups: 1) map-based navigation systems, 2) map-building-based navigation systems, and 3) map-less navigation systems [4]. The concept of the map in most of these approaches is typically on metric

space. Maps are either given by a prior or being constructed online using the sensory information such as LIDAR and RGB-D data [24, 34, 41]. Simultaneous Localization and Mapping (SLAM) [21] is an online automated technique by which exploration and mapping of an unknown environment are done simultaneously. These methods generate high-quality maps with a relative or absolute scale. However, they suffer from calibration/tuning issues, processing time/complexity and dealing with dynamic environments. On the other hand, some other sensory data such as Global Positioning System (GPS) are being widely used by the community for the outdoor-navigation because of its accuracy and ease of use [59, 57]. Tiemann et al in [90] uses ultra-wideband sensor (UWB) for navigation of a UAV. However, these sensors are either needed to be installed in the environment or can be noisy and denied in some indoor environments (e.g. GPS-denied environment).

Mnih et al and Silver et al in [51, 50, 80] showed that deep models trained in an end-to-end manner could predict highly accurate actions by leveraging high-dimensional raw sensory input. Following these works, end-to-end navigation methods using reinforcement- and supervised-learning has gained attraction. Proposed approaches can be categorized as 1) reactive models that are memory-less and navigate based on either semantic features of an environment or based on some other priors on the goal or environment [103, 20], 2) models that are fully trained in an environment and meant to be easily transferred into other environments with localization ability being encoded modeled into the networks' weights [6, 49], 3) models equipped with general purpose memory (e.g. LSTM) or episodic memory [35, 62], and 4) models equipped with a specific navigational memory such map representations [32, 98, 38, 75, 102].

Dosovitskiy et al in [20] trained a reactive model using supervised learning to predict an action based on raw sensory input from a three-dimensional environment. Zhu et al in [103] trained an actor-critic deep model on a high-quality 3D simulator and fine-tuned it on real data. This reactive model predicts an action based on the current scene and the target scene. Training on various scenes makes the model mimic shared generalizable aspects of indoor environments (e.g. TV and couches are usually close to each other). Mirowski et al in [49] proposed a navigational deep model architecture (Nav-A3C) that not only predicts actions but also has depth estimation and loop closure detection as auxiliary tasks. This model relies on stacked LSTM architecture and is trained using reinforcement learning in a complex environment. The policy objective is to explore the environment to find the goal starting from an arbitrary location. If the goal is reached, the agent is re-spawned to a new location and must return to the goal efficiently. The authors believed that adding auxiliary relevant tasks to the model could help build features faster. Although transfer learning can be employed to lower/ease the training procedure for a new environment, Nav-A3C model must be trained well on the target environment since the localization ability of the model is encoded into the weights of the network. Dhiman et al in [14] conducted a set of experiments on [49]'s work to investigate the navigational ability of the Nav-A3C agent under different

scenarios. They have found that when the model is being trained and tested on the same maps, the algorithm can store and exploit map information, but on different sets of maps it fails to transfer this ability.

[32] proposed an end-to-end joint architecture for mapping and planning in which the mapping is driven by the needs of the task. The mapper constructs a metric egocentric multi-scale top-down belief map of the world based on the agent's observations over time. The planner is a value iteration network that takes the belief map as input and outputs the action to take towards the goal. The goal can be specified as geometry locations or as semantic commands such as "Going to a chair." The model was trained on a dataset using supervised learning and tested on both simulated data and the real world. Bruce et al in [6] presented a one-shot reinforcement learning method for learning to navigate to a fixed goal in a known environment. A 3D map was constructed offline after the robot had traversed a path. A deep model was then trained using reinforcement learning based on the samples from the 3D constructed map. The author named the map as *Interactive Reply* memory. Khan et al in [38] proposed a self-supervised policy gradient algorithm for target-driven navigation in an unknown environment. They also employ unsupervised auxiliary tasks as well as a memory augmented value iteration network for the planner. Value iteration network is used as the local planner that takes locally observed space, and the differentiable memory computes globally consistent plans based on a history of local plans over time. Bhatti et al in [3] proposed to augment a standard deep q-learning model with semantic maps instead of going end-to-end and use raw data as input. These semantic maps are aggregated information about the 3D environment and constructed by a standard SLAM algorithm.

Closer to our work, Wang et al in [98] proposed a deep omnidirectional model for visual place recognition. Given a set of omnidirectional images from an environment, O-CNN (Omnidirectional Convolutional Neural Network) can retrieve the closest exemplar and estimate the relative distance between the current image and closest place. A heuristic policy is also proposed which based on the estimated relative distance, can navigate the robot towards the closest place. However, this method does not take into account condition-invariance, and heuristic policy, and is not practical as the robot needs to move around and find each movement has brought it closer to the target place. Oh et al in [56] trained a deep model equipped with an addressable memory using reinforcement learning for navigation. The addressable memory enables the model to have more sophisticated access to the last N frames. Memory operations from these works are similar to MemNN [100] that is from the natural language processing community being used to store the context. However, the architecture is limited to only remembering the past N frames. Parisotto et al in [60] proposed an architecture that employs a spatially structured 2D memory image to store information about the environment over a long time. The model was also equipped with a LSTM and trained using reinforcement learning in a 3D simulator and tested for a *goal-*

search task where the agent needs to remember which goal between two goals it should reach. Zhang et al in [102] approached map exploration using a deep model equipped with an external memory. Exploring a new environment requires a long-term memory, and long short-term memory structure is not sufficient to keep the essential information of the map. The external memory acts as an internal representation of the environment for the model, and SLAM-like procedures are being embedded into the soft-attention based addressing of it.

Savinov et al in [75] proposed semi-parametric topological memory (SPTM) for navigation which consists of a retrieval network for place recognition, a memory graph which is constructed based on the similarity of the representations generated by the retrieval network, and a locomotion network that does the visual servoing given current and target images. Both networks were trained in a supervised manner based on a random agent moving around in a 3D environment. This work looks similar to our approach. However, the retrieval network in [75] is not trained to be condition- and viewpoint-invariant and their experiments and training data were based on a low-quality non-photo-realistic environment. Their memory structure and image retrieval are also different than ours since their work is designed to do navigation while ours is more focused on the visual teach and repeat problem.

3.2 Method

3.2.1 System Overview

The overall structure of the proposed system is similar to semantic VTR, as shown in Figure 2.1 with some minor modification of the YOLOv2 object detector [67] being replaced with PlaceNet and the motion controller being replaced with LocoNet. The core program runs on a stationary computer equipped with a commodity GPU. This program receives video frames from the UAV periodically, passing them through PlaceNet to obtain their embeddings. The rate of the storage is set to be real-time to make sure all visual data are being stored in memory.

In the repeat phase, the UAV motion is actively controlled by the LocoNet. Again, images from the robot are input to PlaceNet to obtain an embedding. The relocalization module then finds the closest match between the current scene embedding and the ones stored in memory. Once localized, the motion controller predicts the best action that would move the robot so that the new scene will look like the next scene in the stored sequence. This action will be sent back to the robot for action. To avoid complication, we used discrete actions in our experiments; we defined six actions: [Forward, Turn Right, Turn Left, Backward, Move Right, Move Left] by which the robot will apply a static amount of movement. Communication between the robot and the stationary computer is over WiFi.

These modules are described in more detail below. The system is implemented in Python3, and our deep learning framework is PyTorch [61].

3.2.2 Environment

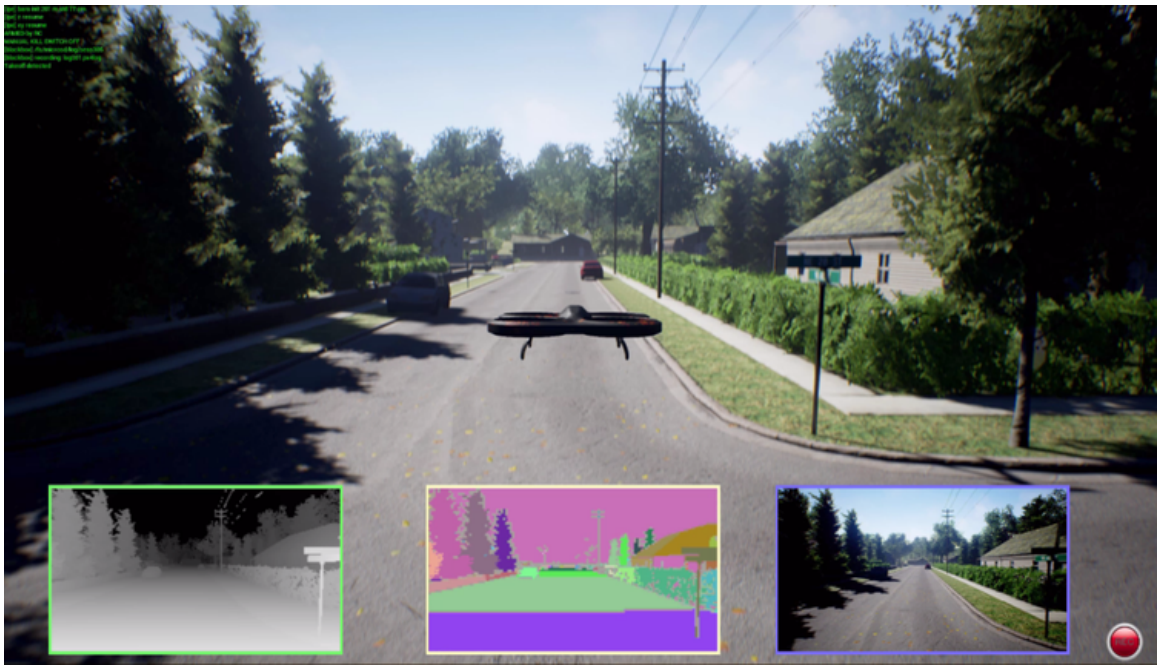


Figure 3.1: AirSim Simulator in a city environment [78] (Copyright ©2018, Springer)

Training a capable deep neural network requires a large set of various training data to make a model generalize better and avoid overfitting. Collecting such a dataset for our models is challenging and the amount of data in available datasets are not sufficient either.

Due to the recent attention to deep neural networks and especially deep reinforcement learning from raw visual data, there has been much effort developing a photo-realistic simulator that can be used to pre-train models to fine-tune them by a few amounts of real-world data afterwards. The gaming industry is moving forward with the aim of photo-realistic graphics too, resulting in commercial game engines that are designed to make game-building easier. Each of these game engines has its market for graphical assets that makes designing the games' environments easier. The robotics community has already utilized this capability and released different photo- and physics-realistic simulators.

AirSim [78] is an open-source photo-realistic simulator for drones and cars. It is built as a plugin on top of Unreal Engine. Unreal Engine is a popular game engine and has thousands of developers all over the world. Using its API, we have access to the robots' sensory data including raw and depth images, global coordinates, and semantic segmentation. We also have control over the environment such as changing the lighting. We chose an existing

environment [15] that is a typical neighbourhood with various houses, cars, streets and trees.

3.2.3 PlaceNet

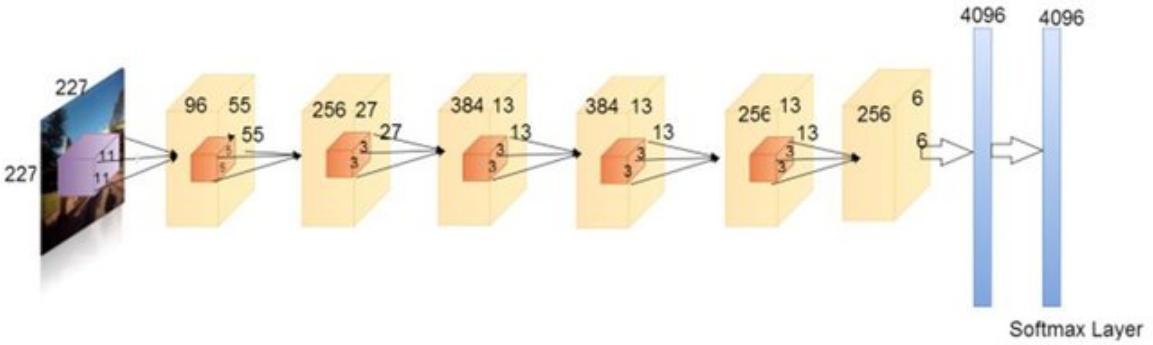


Figure 3.2: PlaceNet architecture proposed by Chen et al. [9] (Copyright ©2017, IEEE)

The network PlaceNet generates an internal representation of an image in a way that the similarity of embeddings generated for two observations from the same scene is high and for two observations from two different scenes is low. This network should generate condition- and viewpoint-invariant features meaning that the embedding of the same scene in different conditions should be close in their feature space. This problem is similar to *Visual Place Recognition* which is a very attractive field of research, and there are tons of approaches proposed for it. With the rise of deep learning, there are attempts to approach this problem by using generic embeddings generated by networks that were trained for other recognition tasks. Although their internal representations would encode visual information of a scene over an objective, they do not necessarily convey salient features that are useful for place recognition and robust to perceptual changes.

Chen et al in [9] provided a massive specific places dataset (SPED) to enable training a multi-scale CNN-based feature encoding to generate condition- and viewpoint-invariant features. The SPED dataset has been collected by images captured from approximately 30,000 outdoor cameras around the world, which have observed the same public place over several years (2007-2017) under different conditions such as environmental changes, lighting variations and season changes. The dataset contains around 2,500,000 images over a wide variety of outdoor scenes. Because there are hundreds of thousands of images in their provided dataset, they avoid training such a huge dataset on a Siamese or Triplet network. Instead, they trained a classification network for this task. Figure 3.2 shows the network architecture in detail.

We adopted the pre-trained Hybrid-Net up to layer Conv4 to transfer conceptual knowledge of the model for our task. According to the paper, the best performance is seen to be in this layer, and it is because features extracted at this level are the most informative and

at the same time abstract and not over-fitted to the dataset. Cosine similarity is used as the similarity measurement:

$$S_{i,j} = \cos(R_i, R_j) = \frac{R_i.R_j}{\max(\|R_i\|_2.\|R_j\|_2, \epsilon)}, \tag{3.1}$$

Where R_i and R_j are the representations (flatten of the output generated by the network at layer Conv4) given image i and j. ϵ is set to 1e-8 to avoid division by zero.

This network is then fine-tuned in a self-supervised fashion by triplet loss. To collect the dataset, a UAV was flown around the environment, starting from an arbitrary position and orientation in each round capturing images at each step. We have discretized the action space for the UAV's motion into six actions as described above. A random action was taken at each step, flying the UAV around for $L = 200$ steps. We collected the data for 1682 rounds, each round starting from a new location. The total number of collected images is 336,263 images. We split the data randomly into three sections. 306,860 data (92%) for training, 6,000 (1% validation, and 23,403 (7%) testing.

Triplet loss take in three samples: an anchor, a positive and a negative. The objective is to make the anchor closer to the positive sample and farther from the negative one. In our implementation, the positive and negative images are selected to be temporally close and far to the anchor respectively in a training traverse as shown in Figure 3.3. Triplet loss is calculated based on the similarity of the anchor and positive image to be high and anchor and negative image to be low:

$$\text{Loss} = \max(S_{a_i,p_i} - S_{a_i,n_i} + \text{margin}, 0) + \alpha * (\|R_a\|_2 + \|R_p\|_2 + \|R_n\|_2). \tag{3.2}$$

Where S_{a_i,p_i} is the similarity between anchor and positive and S_{a_i,n_i} is the similarity between anchor and negative. Margin is a positive number ranging as $0 < \text{margin} < 1$ that forces the anchor-positive similarity be more than anchor-negative similarity. The regularization factor α is set to 0.001.

We trained the model using the Adam optimizer with learning rate $\alpha = 0.001$ and batch size of 64. Learning rate schedule was also used to decrease the learning rate after n=500 epochs by 0.1. We trained the model for 200 epochs and saved the most accurate model on the validation set. We tested the model with the same setting of choosing a positive and a negative for the given anchor, and checked if the similarity score of the positive image was higher. The model achieved an overall accuracy of 79.61% on testing data.

It is worthwhile mentioning that we first tried training this model using a *Siamese* architecture [40]. In this architecture, both current and target observations are fed into an encoder network to generate embedding vectors for each. These resulting embeddings are concatenated and processed by a fully-connected network to generate similarity probability. This small fully-connected network was trained using cross-entropy loss (typical classifica-

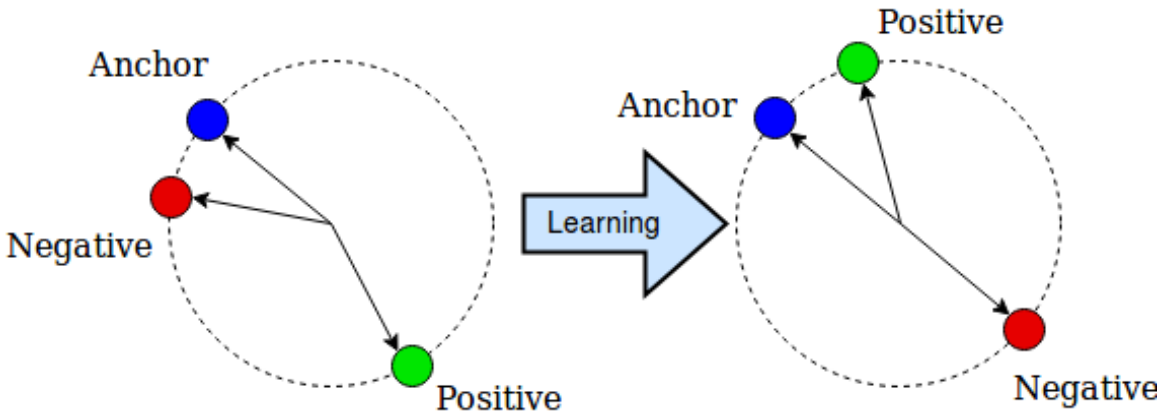


Figure 3.3: Triplet loss in cosine similarity.

tion loss) in which given (anchor, positive) and (anchor, negative) the output classes are 1 and 0 accordingly. Although this network shows acceptable results on the validation set, the processing time needed to compute a similarity score for each pair of current image and reference memory is high, resulting in a bottleneck on similarity matrix calculation. For this reason, we have moved on to the triplet loss and cosine similarity measures which were faster than the fully-connected network.

3.2.4 Reference Memory

PlaceNet model provides an embedding as a feature tensor of size 4096. This embedding encodes the salient information of the input image, that is useful for place recognition; meaning that the network would ignore dynamic objects of the scene such as a *person*, a *cat* as their world positions are not fixed. We pre-compute and store embeddings generated by PlaceNet given the images from the teaching phase. In an ideal scenario only the embeddings are stored in reference memory with their size being comparable with the size of the descriptors in semantic VTR. However, LocoNet take in raw images and is completely separate from the PlaceNet. For this reason, we had to store raw images as well as the embeddings. This issue can be resolved later by designing and training this system in an end-to-end manner with only one model responsible for both place recognition and short-range navigation.

3.2.5 Relocalization

In the repeating phase, the relocalization module first finds the closest match between the current scene and the pre-computed embeddings of teaching observations stored in reference memory. It then selects the best desired waypoint to be given to the navigation module.

In the first step, similar to our semantic VTR approach on relocalization, we employ the SeqSLAM technique [47] where in localization is done over a contiguous sequence of

embeddings. Following the paper, at each iteration, we compute the embedding of the current scene and store it in a sequence with size d_s: $\{R_{T-d_s+1}, ..., R_T\}$. Each of these embeddings are compared with the embeddings stored in reference memory as per Equation 3.1 to create a similarity matrix as drown in Figure 2.4. Since the robot may have different velocities in the teach and repeat phases, we must test different velocities to find a good match. We employed the same temporality constraint as we had in semantic VTR:

$$M = \begin{bmatrix} S_{T-d_s,0} & S_{T-d_s+1,0} & S_{T-d_s+2,0} & \cdots & S_{T,0} \\ S_{T-d_s,1} & S_{T-d_s+1,1} & S_{T-d_s+2,1} & \cdots & S_{T,1} \\ S_{T-d_s,2} & S_{T-d_s+1,2} & S_{T-d_s+2,2} & \cdots & S_{T,2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ S_{T-d_s,n} & S_{T-d_s+1,n} & S_{T-d_s+2,n} & \cdots & S_{T,n} \end{bmatrix}. \tag{3.3}$$

Where $M_{(d_s \times n)}$ is the similarity matrix and $S_{T-ds+2,4}$ is the cosine similarity between the third observation in sequence and the fifth element in reference memory.

$$S_i^v = \sum_{j=0}^{d_s} M_{k,j} \tag{3.4}$$

Where S_i^v is the trajectory score for index i with velocity v which ranged between V_{min} and V_{max}, k is the particular representation in reference memory that the trajectory passes through:

$$k = \max\left(i - (v * j), 0\right) \tag{3.5}$$

After initial relocalization, we positively weigh descriptors nearby in the sequence to reduce noise and make relocalization smoother. This also helps to handle time-extended loop-closures where the taught trajectory has repeated sections and may cause a robot to get stuck in a loop. The more confident the previous match, the narrower is the distribution we weigh nearby descriptors with:

$$\hat{S}_i^v = \frac{S_i^v(1 + \beta \cdot \mathrm{norm}(x = i, \mu = i', \gamma)}{(1 + \beta)} \tag{3.6}$$

Where $\hat{S}_i^v$ is the score of matching that has been done by sequential comparison for index i and velocity v. β is the factor of temporality. β is selected by the confidence of the previous match. i is the index of the memory which is being calculated and i' is the average of the last five matched indices of the memory. I_{match} is calculated by finding the maximum score over all the indices and velocities:

$$I_{\mathrm{match}} = \max_{i \in [0,n]} \left\{ \max_{v \in [V_{\mathrm{min}}, V_{\mathrm{max}}]} \{\hat{S}_i^v\} \right\} \tag{3.7}$$

In the second step, we retrieve the desired waypoint to be passed to LocoNet. However, selecting a target close to the match causes the robot to make no progress whereas a distant target causes LocoNet to fail at taking the proper action. This depends on the velocity of the camera at different stages of the teaching phase. We implement a look-ahead strategy to find a reachable and progressive waypoint. This is done by computing the probability of the actions generated by LocoNet given the current scene and closest match and following frames up to a fixed length. We select the last waypoint that has a probability above the chosen threshold to assure reachability, and make progress:

$$I_{\text{desire}} = \max_{i \in [I_{\text{match}}, I_{\text{match}}+D]} \{\text{LocoNet}(O_{\text{current}}, O_i) > P_{\text{threshold}}\} \tag{3.8}$$

Where D is a fixed length that is set to 50 in our implementation. $0 < P_{threshold} < 1$ is a fixed similarity threshold that ensures reachability. Based on our experiments we set it to 0.85. The outputs of the relocalization module are $O_{I_{match}}$ and $O_{I_{desire}}$ that is the closest observation and the desired waypoint accordingly.

3.2.6 LocoNet

The network LocoNet predicts an action given current and target observations, by taking which the robot will see the target image. In literature, this problem is known as *Visual Servoing*. We have already mentioned that the actions are discretized to six: [Forward, Turn Right, Turn Left, Backward, Move Right, Move Left]. LocoNet generates action probabilities using softmax. An action can be chosen by either choosing the most probable one or by sampling from a distribution. We use the most probable action with a heuristic to avoid taking an action that will cancel out the previous action. For instance, if the previous action was $Forward$ and now the most probable action is $Backward$, we choose the second most probable action. This condition is known to happen because of the discretized action-space, that mean a target observation may not perfectly match the new observation made by the robot upon performing an action. The robot could potentially be stuck in a loop of $TurnRight$ and $TurnLeft$.

We use ResNet-18 [33] as LocoNet architecture with the input size $224 \times 224 \times 6$. This network is trained in a self-supervised fashion based on the dataset collected for place recognition using AirSim simulator. Table 3.1 shows the number of data and its breakdown per class for training, validation and testing modes. The dataset contains a set of observations and actions taken at each step as: $[O_0, O_1, O_2, ..., O_{n-1}]$ and $[a_0, a_1, a_2, ..., a_{n-1}]$. The amount of movement/rotation for an action is an arbitrary number for generalization purposes. We generate training samples by pairing two temporally close observations O_i and O_j in the dataset by a random distance of most 20 index unit with an additional heuristic on the opposite action. We choose a temporally close pair that does not have a different action till then. Cross-entropy loss is calculated based on the pairs and the actions taken as:

$$\text{Loss}(x, \text{class}) = -\log\left(\frac{\exp\left(x[\text{class}]\right)}{\sum_j \exp(x[j])}\right). \tag{3.9}$$

The Adam optimizer is used in our implementation with a learning rate $\alpha = 0.001$ and a batch size of 64. The learning rate schedule is also used to decrease the learning rate after n=500 epochs by 0.1. This could help the optimizer start with a more global search and end in local optimal. We trained our LocoNet with 200 epochs and saved the most accurate based on the its performance on the validation set. Table 3.2 shows the precision, recall, and f1-score for LocoNet, running on the test data. It achieved an overall accuracy of 61% on testing data. This suggests that the network has difficulty on estimating the correct class for $TurnRight$ and $MoveRight$ actions as well as $TurnLeft$ and $MoveLeft$ actions, since these classes are visually closer than that between $MoveForward$ and $MoveBackward$.

We also tried training the model using reinforcement learning. Given a short path started from an arbitrary location and navigated to by a wanderer agent, we re-spawn the agent to its initial location and force it to repeat the path. We tried both Q-learning [51] and Actor-Critic networks [50] with different reward functions and training processes. However, training such a huge model using reinforcement learning from scratch did not seem to be a reasonable solution, since our objective could be done by supervision. For this reason, we will skip the details on implementation and analysis of our reinforcement learning method.

Mode	Forward	Backward	Turn Right	Turn Left	Move Left	Move Right
Training	49,593	49,407	49,379	49,354	49,660	49,573
Validation	1,000	1,000	1,000	1,000	1,000	1,000
Testing	3,795	3,944	3,918	3,869	3,939	3,938

Table 3.1: Number of class data in the training, validation, and testing modes.

Class label	Precision	Recall	F1-score	Support
Forward	0.65	0.55	0.60	3795
Turn Right	0.55	0.69	0.61	3944
Turn Left	0.53	0.76	0.63	3918
Backward	0.69	0.51	0.59	3869
Move Right	0.63	0.54	0.58	3939
Move Left	0.63	0.55	0.59	3938
avg/total	0.61	0.60	0.60	23403

Table 3.2: Precision, recall and F1 score for LocoNet on test data.

Figure 3.4: Test environment where we pilot the robot around a block in AirSim (trajectory in red) from starting location colored in yellow.

3.3 Experiments

We performed experiments in the AirSim simulator and postponed real-world experiments since our models were only trained in the simulator. Using a photo-realistic simulator is beneficial and eases the process of fine-tuning with real-world examples. But this alone is not sufficient to perform well in real-world scenarios. AirSim provides us with real-time information including position and orientation of the robot, that enables us to conduct our experiments and analyze teaching/repeating trajectories. The test environment is pictured in a schematic in Figure 3.4.

3.3.1 Training

In the training phase, the UAV is piloted over the trajectory shown in Figure 3.4 using a set of pre-computed actions. Each action performed at different velocities, meaning that our recording rate is dynamic and more similar to a real-world scenario. The altitude is constant at 6m. The pose is recorded by AirSim API for further analysis. The video is then fed to the PlaceNet network to generate embeddings, a total number of 220 keyframes are stored in the reference memory. Similar to our first chapter on semantic visual teach and repeat, we have conducted different cases to test the robustness of the system.

Figure 3.5: A sample keyframe from the AirSim environment during the teaching phase.

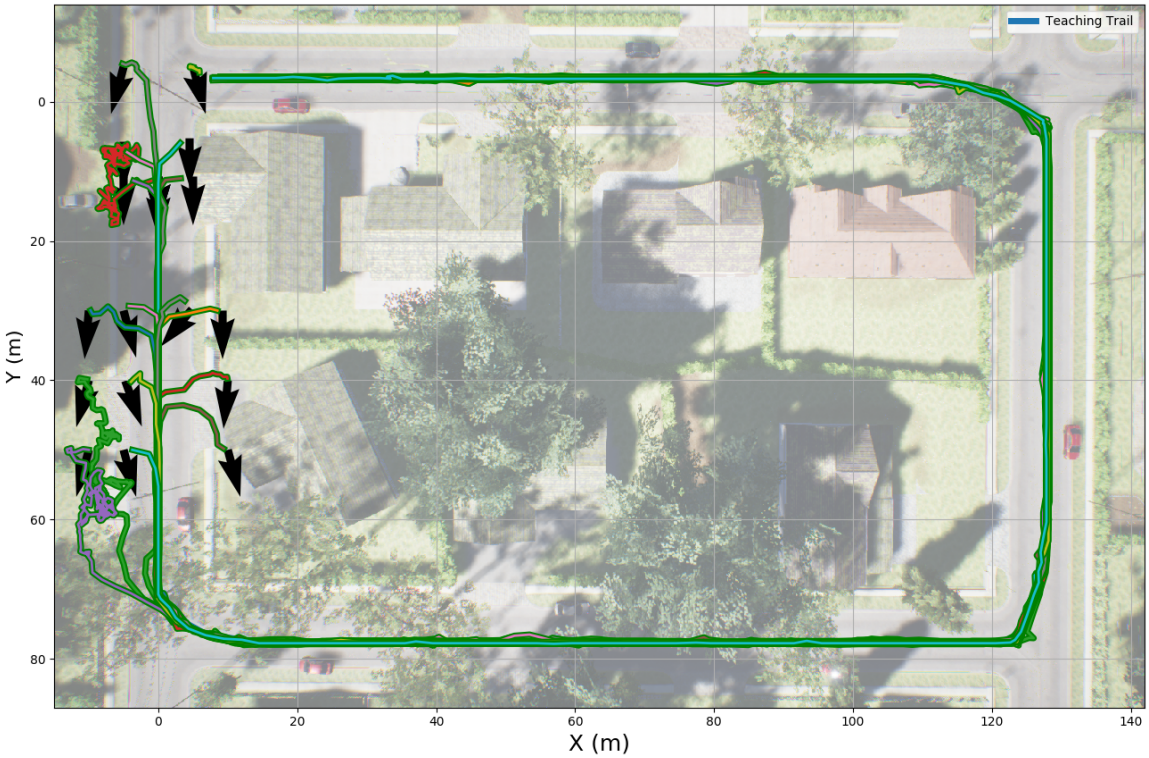
3.3.2 Experiment I: Changing Viewpoint

A first experiment was performed to test the robustness of relocalization to changes in initial camera position and viewpoint. We run 15 trials with the robot starting from (x, y, θ) poses chosen at random up to $10m$ away from the original taught location as well as different altitudes chosen at random. As described in the method section, it will relocalize itself within the taught path and plans an action accordingly, in case of relocalization failure it rotates on the spot until relocalization is achieved.

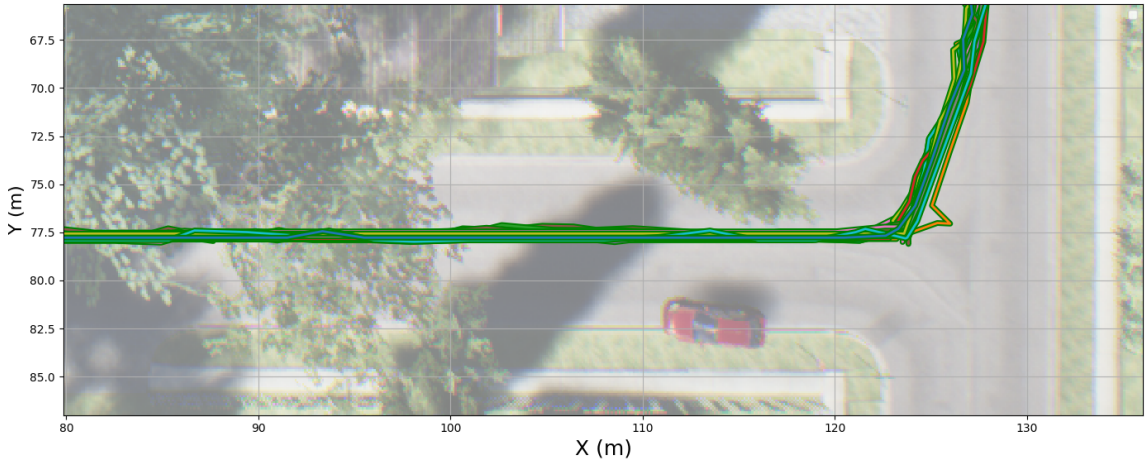
Results

In all 22 trials except one, the robot has repeated the path successfully, arriving less than 0.5m from the original end-point. The only failed trial, initial location was behind a tree, and all the waypoints were out of the robot's vicinity. As a result of adding lateral actions $MoveLeft, MoveRight$, robot converges to the taught path instantly and repeats the path afterward. Figure 3.6 reports the start positions and the paths of all cases on a transparent top-down view of the map. All successful trajectories are close so that minor errors are not visible in the figure. The bold blue line is the taught path.

Running experiments, we have found out that the system is vulnerable to different altitudes, meaning that the robot can not repeat the path from a different altitude (around 2m up/down) than the teaching altitude. A reasonable cause that can explain this incident is that our machine learning model has not been trained to navigation with different altitudes. We can either add $MoveDown, MoveUp$ actions to model this behaviour or train our model with data in distinct altitudes.



(a) Full-view



(b) Zoom-in on a piece of trajectory

Figure 3.6: Trails in both teaching and repeating phases during experiment I, arrows are indicating initial rotation of the robot. Transparent background is the top-down view of the map.

Also, comparing the results of this experiment with the semantic VTR version, we have found out that our locomotion module is far stricter. One explanation is the type of data that semantic VTR uses to plan an action, object detections and bounding boxes are too abstract by which they ease the motion controller's constraints. Even though the upcoming

frame does not match with the desired waypoint by doing a planned action, but the robot is making progress and goes towards the target. On the other hand, LocoNet is trained to navigate the robot from one frame to its nearby target frame. So that we can not expect the agent to navigate when the target is out of its vicinity, this goal can be achieved by training a model to predict six degree-of-freedom camera relocalization.

3.3.3 Experiment II: Changing Environment

This experiment investigates the ability of the system to repeat a taught path when the environment changes. In this experiment, the robot starts approximately at the same place as in teaching. In one set of trials we remove/moved 30 objects that were significant in the environment such as moving all cars on roads, as well as some trees alongside roads. In the second set of trials we change the lighting of the environment by changing the position of the sun. AirSim simulates the movement of the sun based on the day of the year. We ran a trial every hour in 24 hours to cover the entire day.

Results

Examining robustness to appearance changes in the environment, we record the time taken to repeat the trajectory after (a) simulating the sun movement during a day and (b) removing/moving objects in the environment. A histogram of flight times is shown in figure 3.7 with fitted Gaussians.

By removing/moving 30 objects system showed the same performance as the fitted Gaussian in 3.7 is comparable, Figure 3.8 also shows that robot has successfully repeated the trail in all 30 trials, moving precisely like the teaching phase. However, trying an extreme version of the experiment with more than 60 critical objects removal/modified, it failed at ending the path. One valid explanation is the lack of training data with modified and dynamic objects. This can be improved by collecting data in a dynamic environment where cars are moving, and trees are shaking in the wind.

Changing the illumination by simulating the sun movement, all the trails that were close in the teaching time have successfully repeated the trail completely. Teaching time was around noon time, seven successful trials were from 10 am to 4 pm. However, trails on 3 pm and 4 pm finished the trail in a longer duration as shown in 3.7. All the trials are shown in Figure 3.9, system is not performing good in different lighting situations. Especially at the start of the trail where some high trees and houses cause severe changes in the scene. Although the PlaceNet performed well in finding the closest match and target waypoint because of its training data, LocoNet was unable to predict the correct action. Sun position and shadow change the scene extremely, and this performance was predictable. We trained our LocoNet under same environment condition such as lighting and changes in its appearance. Collecting data in different conditions was impossible due to the currently available APIs that AirSim provides.

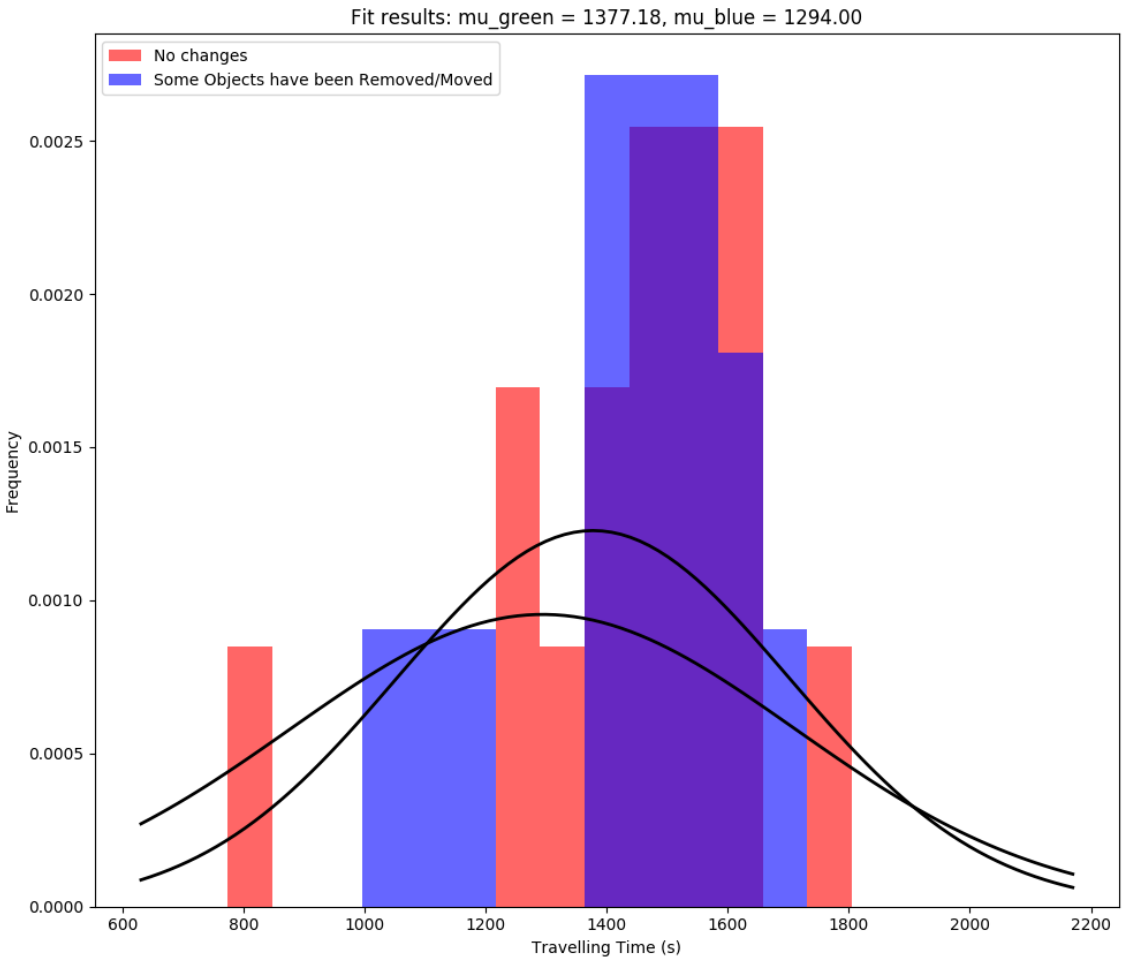
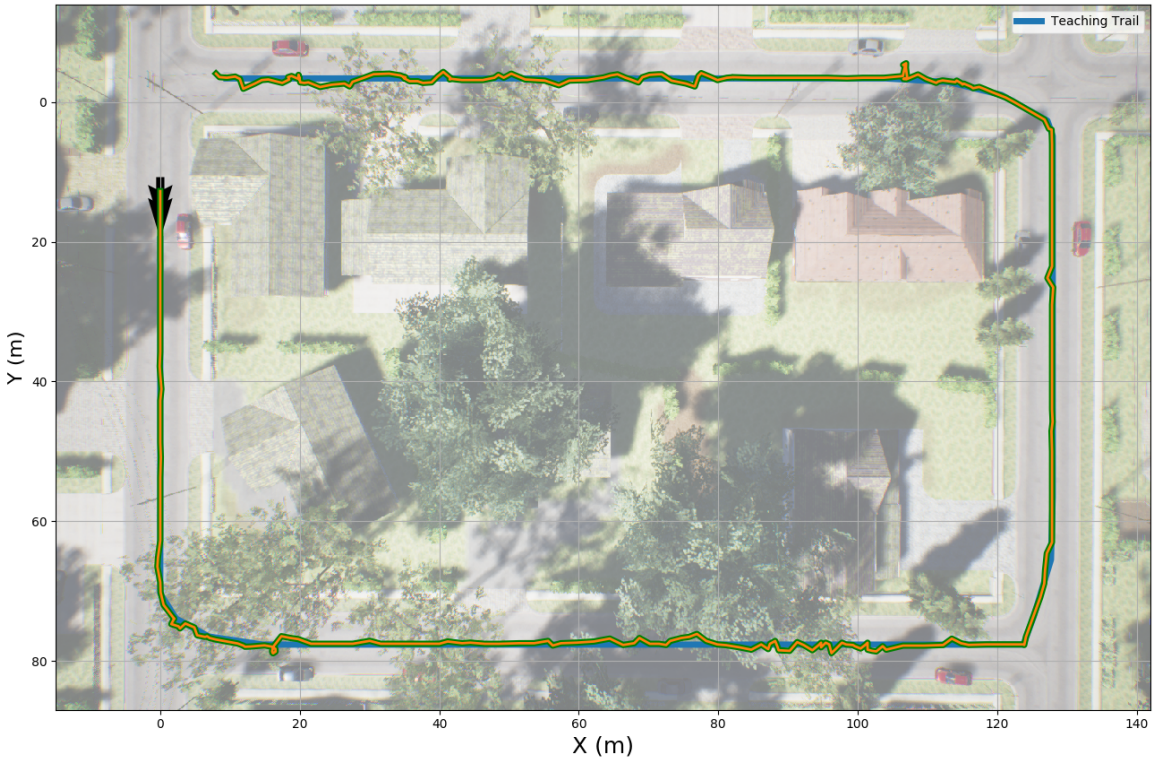


Figure 3.7: Repeat completion times for identical environment (green), light-changed environment (red) and object-moved environment (blue).

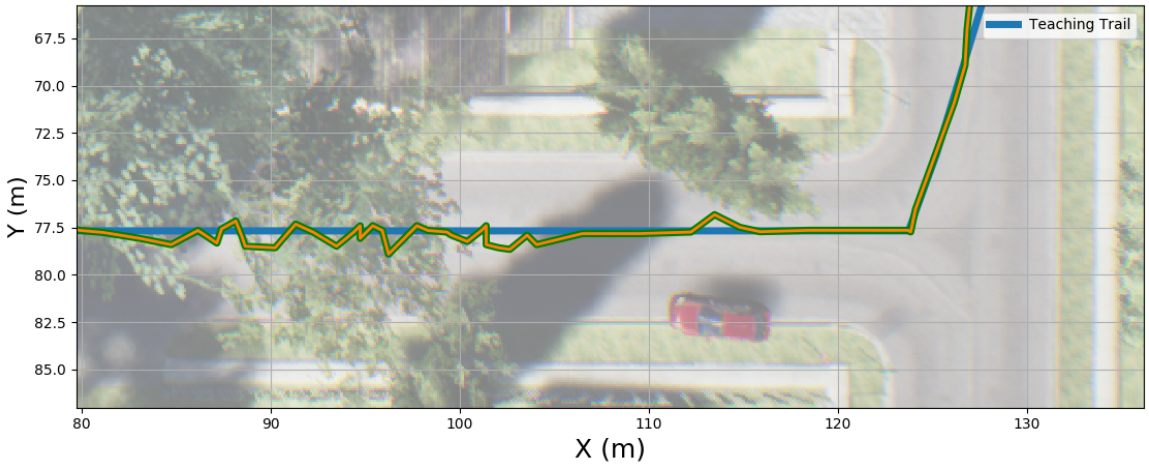
We are looking forward to having AirSim with these APIs by which we can train our models in a more natural and realistic situation, having dynamic objects and manageable lighting.

3.4 Conclusion, limitations and future work

In this chapter, we propose a new strategy for the task of visual teach and repeat by employing deep models that are capable of extracting the crucial features of a scene required for place recognition and navigation. In the second chapter, we show that highly meaningful semantic objects can be utilized to repeat a trajectory effectively. However, in a real-world scenario, it may fail due to the lack of semantic objects visible in a scene. By employing deep models, our system can use both low-level and high-level features in a scene and be robust to different conditions. However, training such a model requires a diverse and comprehensive



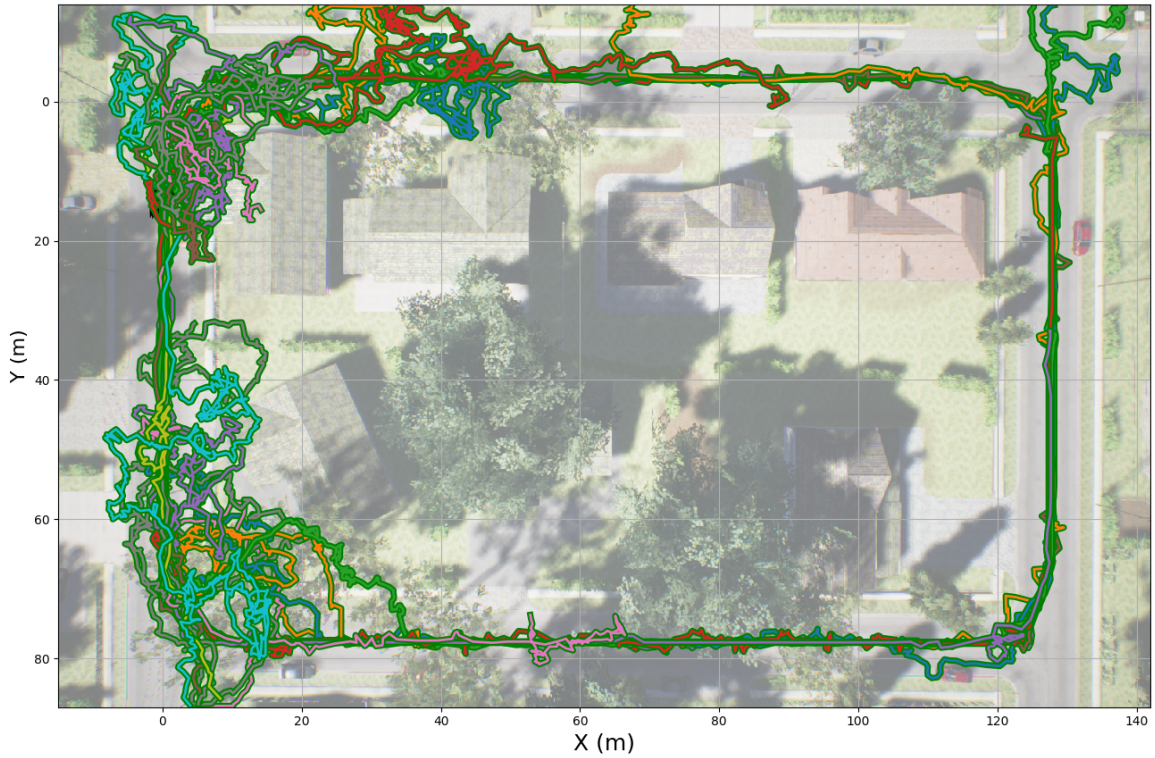
(a) Full-view



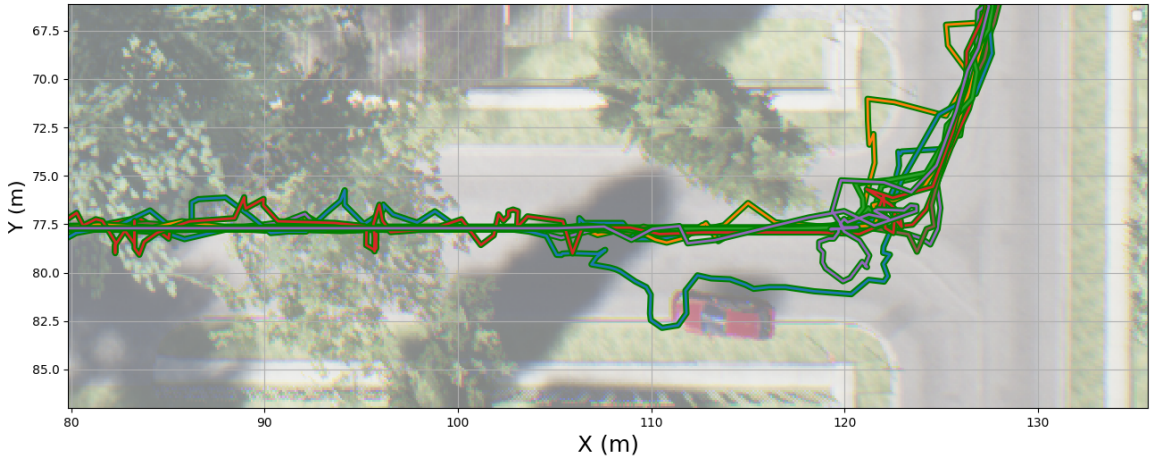
(b) Zoom-in on a piece of trajectory

Figure 3.8: Trails in both teaching and repeating phases during experiment II with removal/changes in the environment. Transparent background is the top-down view of the modified map.

training set. Collecting such a dataset is challenging and time-consuming. Instead, by using modern photo-realistic simulators, this problem can be solved. We can pre-train a deep model in a simulator and then fine-tune it using a few sets of real-world training data.



(a) Full-view



(b) Zoom-in on a piece of trajectory

Figure 3.9: Trails in both teaching and repeating phases during experiment II performing during different times a day.

In this work, we use AirSim simulator; it is developed based on the Unreal game engine framework and is still under development to add more features and API accesses.

We test our method in the simulator because it was not trained by real-world data. We use an environment akin to a typical neighbourhood provided by the Unreal community. We demonstrate that the robot can start far away from the taught initial position, facing

at a random heading, and the robot relocalizes itself and repeats the taught trajectory. We also showed the robustness of the method to minor changes in the environment.

Our method depends highly on the deep models, and deep models' performance depends on the training data. A diverse and comprehensive dataset makes a deep model to generalize better and extract highly informative features such as textures and salient objects. PlaceNet model showed a good performance under different conditions as it was trained by a very diverse and general dataset. However, we trained LocoNet from scratch using simulation data we have collected. Although it showed a good performance on repeating long trails, it was sensitive to extreme changes in the environment. We are planning to collect a more diverse dataset in the future by the time that AirSim provides a sophisticated access to the environment. We are also planning to fine-tune the network by real-world data such as SFU Mountain Dataset [7].

Chapter 4

Visual LOST: Visual Localization-Space Trails for robot teams

In this chapter, we propose a method for target-driven visual navigation tasks in an initially unknown environment using a team of robots. This work is motivated by the application of cooperative pathfinding using ant-inspired trail following.

Ants form chemical trails connecting food sources to the nest which enables them to harvest food sources and navigate through complex environments. Humans can search for food, return home upon finding it and share the path information because of their complex brain which is capable of visual and linguistic abilities. However, ants with their simpler brain communicate with each other using chemical scents known as pheromones when foraging. They use pheromones in two ways: 1) finding the way back to the nest upon discovering a food source. 2) informing other ants about the food source's location. Each wandering ant leaves a trail of pheromone while looking for food sources. Upon finding a food source, it follows its trail back to the nest while laying more pheromones on the ground. This makes its trail stronger and thicker. Meanwhile, when other wandering ants find a chemical trail, they give up on searching and start following the trail. There may be more than one trail to a food source from the nest. Deposited pheromones on the ground evaporate from the weather, temperature and sunlight, meaning that pheromones on longer paths evaporate faster. This feature enables ants to converge to the shortest route discovered so far. It also makes them robust to changes in the environment such as trail blockage by an obstacle or the failure of many individuals.

LOST [96] examined an ant-inspired trail-following method that enables a team of imperfectly-localized robots to navigate between places of interest. The authors showed that the method "1) copes with accumulating odometry error; 2) is robust to the failure of individual robots; 3) converges to the best route discovered by any robot in the team". LOST uses waypoint coordinates as landmarks. They are generated online by each robot and

shared with other teammates. Waypoint coordinates can be in any shared localization-space. Localization-space is “any consistent spatial or topological representation of position", and shared localization-space is when “there is some correlation between representations maintained by two or more individuals." One example of shared localization-spaces is two robots starting from one initial location in the same coordinate system maintaining position estimate via odometry.

Here we will examine deep visual features as a shared localization-space following a LOST approach to the problem. Consider a team of robots all equipped with RGB-D cameras and connected to a radio network which enables them to share information. These robots will employ ant-inspired emergent trail following first to find their target and then converge to the best-discovered home-target path. Each robot generates visual waypoints based on landmarks while foraging, and shares them with other robots. These visual waypoints are representations generated by a deep model given images captured by each robot's camera. They are reference-independent; there is no need to have a fixed reference frame to have shared localization-space among all robots. They do not have cumulative drift and are individually informative enough to be used for localization, unlike other methods like Odometery (wheel or visual) which may drift over time because of cumulative error. The Global Positioning System (GPS) is a geo-reference localization method where the reference point is fixed among all GPS devices; there is no need to settle on a reference frame. In this work, we are considering environments where GPS signals are weak or unreachable.

We discuss work related to deep learning approaches in cooperative and non-cooperative pathfinding in section 1. Our method and system design are presented in section 2 followed by experiments in section 3. We conclude by discussion and possible future work in section 4.

4.1 Related Work

Dorigo and Caro [17] first introduced Ant Colony Optimization (ACO). The authors put previous algorithms [16, 19, 18] inspired by the foraging behaviour of ant colonies in one framework and defined it as Ant Colony Optimization. Since then, researchers have proposed methods inspired by ant-foraging behaviour for a wide variety of optimization problems such as transportation, load balancing and routing [81, 83]. It has also been applied to a set of well-known robotics problems such as path optimization, multi-agent navigation, and more [52, 26].

[79] proposed a simple neural network model for ant chemical trail following in both virtual simulation and the real world. Alcohol was used as the chemical as it has similar properties to ant pheromone such as evaporation and sensibility. In [70, 71], a robotic ant was built and programmed with a trail-following algorithm. The robot was equipped with an odour pen for trail-laying and a pair of odour sensing “antennae". Svennebring et al in [88]

employed a trail-following algorithm for terrain coverage using a big black pen for trail-laying and infrared proximeters to detect the marks. There is also recent work [46, 93, 27] which used ethanol, gas, and phosphorescent as pheromones to mark the environment. However, the use of chemicals or physically marking an environment is typically impractical in real-world scenarios.

Other than chemical trails, virtual pheromone trails have also been used for trail-following algorithms. Payton et al in [64] used virtual pheromone trails which are being transmitted peer-to-peer via infrared messages. Vaughan et al in [95] employed a trail-following algorithm in a global localization-space inspired by ant foraging tested on a simulator. Agents were equipped with a GPS-like coordinate system; enabling them to share their position in the global coordinate system. The authors stated that the method is robust to significant noise imposed on coordinate data. Later on, [94] implemented the same strategy on real robots with odometry integration as the localization method. Having all robots starting from one location successfully aligned those robots' coordinate systems in a global localization-space. However, odometry is not entirely accurate, and these small errors accumulate by integration, resulting in enormous drifting over time and a diverged localization-space.

Vaughan et al in [96] expanded the work and designed a system that uses a self-referenced, non-stationary coordinate system. Places (nest, food) have fixed positions and are known to robots. This is a factor that can be utilized to fix the drifting problem in rate-measuring sensors. Each robot applies a transformation on the incoming trails based on the places that are known for all of them. [96] also listed a set of constraints that a localization-space must satisfy to become applicable. One of the main constraints that we focus on in this chapter is that the localization-space must be good enough to enable the robot to make a single traverse. For odometry and other rate-measuring sensors, this could become a problem as the integration error/drifting may become so severe (in very long paths) that the system fails to make a single traverse. [96] has been expanded over time by the community to overcome scalability issues but no work studies other localization-spaces that this system could be applied to. Sadat et al in [73] restricted the sensor field of view to make robots lose track of the trail and look for new trails towards the goal. Eventually, they will create new trails connecting places, and the system can handle more agents traversing among places. Sadat et al in [72] also studied scalability issues of the original work where robots were facing each other and blocking the way. The authors proposed a spread-out behaviour for robots to shift home-food and food-home trails apart from each other. Sadat et al in [82] has proposed another shifting behaviour regarding multi-objective multi-colony system where the trails are intersecting each other and causing a blockage. Abdelaal et al in [1] has expanded [72]'s method for the case of having the both to-home and to-food trails flooded with robots and proposed separate trails (called highways) for both paths.

Localization in metric space is understandable and natural. A trail with a set of metric positions lets us imagine the trail from a top-down view 2D map of euclidean space. However, appearance-based localization (as we will discuss later in this chapter) does not have the same perspective. The distance is not in the metric scale and is more about the similarity of scenes. *Appearance-based SLAM* and *Visual Place Recognition* are popular topics in robotics and computer vision in which these issues have been studied [44]. Cummins et al in [13] proposed FAB-MAP (Fast Appearance-Based Mapping) as an approach to appearance-based SLAM. Common objects in an environment appear to have more common appearance words (features descriptors). Based on this fact, a generative model was trained on a bag-of-words dataset. Milford et al in RatSLAM [48] has studied the hippocampus of rodents and proposed a simultaneous localization and mapping model inspired by it. The model integrates odometry with landmark information and forms a consistent representation of the environment. Glover et al in [30] proposed the hybrid RatSLAM/FAB-MAP model and showed that it could perform well even in a difficult outdoor condition. Milford et al in [47] proposed a method to take into account the spatial information of a path and also to overcome the perceptual-aliasing problem. The method considers a set of images (current image to N previous images) to be compared to the memory data instead of comparing just only one image. The distance function was based on a normalized sum of pixel intensity differences. Later on, with the rise of deep learning, visual place recognition and appearance based approaches using deep models gained attention due to their ability to extract high-level features from raw visual data. Sunderhauf et al in [85] proposed a system that uses an object proposal technique to identify potential landmarks within an image and extracts robust high-level features from each proposal by a deep model that is condition- and viewpoint-invariant. Chen et al in [10] fine-tuned a convolutional neural network model that was pre-trained on a visual recognition dataset, and added spatial continuity based on [47]. Chen et al in [9] expanded the work and proposed a multi-scale CNN model trained on a massive specific places dataset (SPED). This model was able to generate condition- and viewpoint-invariant features. The dataset was collected by images captured from approximately 30,000 outdoor cameras around the world, which have observed the same public place over several years (2007-2017) under different conditions such as environmental changes, lighting variations and seasonal changes.

4.2 Method

In this section, we first introduce deep CNN features as a shared localization-space, then we discuss the LOST approach to the problem of pathfinding along with its definitions and constraints. Finally we apply our localization-space to LOST after redefining some of those definitions and constraints.

4.2.1 Visual Localization-space

We formulate a position in space by three attributes, namely, CNN features of the image, Camera Pose, and Field of View. For example, two images taken from two closed positions with the same field of view look similar, especially with possible conditional changes such as lighting or illumination. One of the issues regarding this localization-space is perceptual aliasing; there are many different places that can still appear very similar. However, we can overcome this problem by considering sequential/temporal factors.

Animals use visual landmarks to navigate their surroundings. Although environments can look different depending on the lighting condition and from changes over time (as some of the landmarks may disappear or new ones may appear), animals can still effectively recognize their environments and navigate through them.

PlaceNet was pre-trained on the SPED dataset [9] and fine-tuned on simulator data, while LocoNet was trained from scratch using simulator data. PlaceNet generates embeddings of a given image such that they can be used for relocalization, while LocoNet commands the robot toward a goal given the current image taken from the robot and the goal image

In the previous chapter, we showed that by employing a deep convolutional neural network, we could accurately relocalize the agent within an environment and command it to repeat a long path it is taught during the teaching phase. We showed that PlaceNet and LocoNet are condition- and viewpoint-invariant if they are trained well on a vast dataset containing many different conditions. They do not suffer from drift, requirement of calibration and a fixed reference-frame.

4.2.2 Definitions

Events

LOST defines an event as a “task-relevant occurrence that is perceived by a robot." In a food transportation task, *picking-up food* and *dropping-off food in the nest* are the relevant events. We will be using this definition as well. For simplicity, we are only considering finding a visual target in an environment in our experiments; robots are only supposed to find the goal without the requirement of returning home and dropping food. Upon finding it, the agent re-spawn from the initial location and tries to reach the goal again; basically, we have only one event: finding a given visual target. Based on the equation below we can compute the similarity score between the current frame captured by the robot and the given target image. Similarity score higher than the given threshold means the robot reached the goal.

$$reach(R_{current}, R_{target}) = \begin{cases} 1, & S(R_{current}, R_{target}) \geq T_{target}. \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

Where $R_{current}$ and R_{target} are the current and target representations. S is the similarity function. In our implementation we used cosine similarity as the distance function for both training and testing networks. T_{target} is the threshold, we chose 0.95 based on our experiments.

Places

One of the main challenges of sharing information about places of mutual interest in a group of robots is the requirement of having a global coordinate system or a general map. LOST's main contribution to this problem is to remove this requirement by using the commonality of *Events*. Although each location in space has different coordinates for each robot (because of the robots' frames of reference and drifts in their coordinate frames over time), it has a permanent location in the real-world. This property can be used to fix drift and solve the shared coordinate frame requirement. In the food transportation task, *picking-up food* happens at place A and *dropping-off food in the nest* happens at B. Since robots navigate between these two places only, one robot's trail would normally be transformed into another robot's coordinate system using a transformation matrix (translation, rotation, scale). However, in our approach, positions are reference-free and self-descriptive, so there is no need to do a transformation. A place was represented as a name and a position in localization-space in LOST, which in our approach is an image.

Crumbs

Crumbs or local waypoints in a trail are data structures that contain information about the trail in which they are located. Each crumb has a place P to which it refers, a position L in localization space, an estimated distance d (in time) from the current location of the Crumb L to place P, and a time t when the Crumb was created. This is akin to an ant's pheromone by which they mark a trail to a food source. Based on our proposed localization-space, deep CNN features are stored in crumbs as their location L, in contrast to the original LOST experiments, where metric locations are estimated via integrating odometry. We chose time as the distance value since turning-in-place left or right may cause the robot to get closer to the goal in a trail but does not move the robot in space.

Trails

A Trail consists of a set of crumbs and places. At the start, all robots start with an empty trail, and they add new places and crumbs over time based on the trail-laying algorithm. In the original paper, once an event occurs at a location, that place will be added to the trail. All existing places that are located at the same event are deleted. LOST proposed this idea of having the most up-to-date location of a place in the trail. The reason for doing this is that the position of a place in the localization-space may drift over time for each

robot and by having the most recent location robots can share their trails and transform their coordinate frames based on the transformation of the places. If two trails have two places in common, they can be combined by a transformation that maps positions of two places from one trail to another. In our proposed method, there is no need to keep the latest position and doing a transformation to combine two trails. Since a position in our image-based localization-space is merely a picture taken from that location, it does not drift over time. However, considering a dynamic environment where objects may get moved or removed over time, we keep the most recent position to handle these situations.

Trail decay

In ant foraging systems, trails evaporate over time because of heat and sunlight, which causes the colony to converge to the most optimized path discovered yet. To simulate this, crumbs older than an age threshold is destroyed. Another outcome of this behaviour is that it handles changing environments; fresh waypoints capturing new information will be added while the old ones get evaporated.

4.2.3 Constraints

LOST has a set of constraints that must be satisfied by any suggested localization-space. Here we list those constraints and discuss how our proposed localization-space handles them.

1. “Robots must maintain a localization estimate": Each robot can relocalizes itself in the trail using cosine similarity between the current image and crumbs. This is akin to Euclidean distance in metric space. In the previous chapter, we showed that our method could accurately do relocalization and it is condition- and viewpoint-invariant.

2. “Localization must be good enough to enable the robot to make a single traversal": This means that the localization error in a single traversal between events should be small. For example, using odometry in a long path may fail this constraint since the accumulated drift can be significant. However, visual localization-space is reference-free and self-descriptive, and as we showed in the previous chapter, it can repeat a long taught trajectory efficiently.

3. “Robots can only share trails about places they mutually recognize": This is a general constraint that is not related to the localization-space.

4. “Events occur in fixed places": This is a general constraint. However, in the case of dynamic places, all robots will fail to reach the goal event at the end of the trail, and after a while when all the crumbs are evaporated, they will search for a new goal event as there are no crumbs to follow.

5. “Robots must have some search strategy when reaching the end of the trail to recover from accumulated trail-shortening": In order to recover from errors such as localization errors or to reach the end of the trail but not receiving the goal event, a robot should keep

searching for the goal. This is also a general constraint that is not related to the localization-space. However, we have implemented a searching behaviour for our localization-space that we will discuss further in the experiments section.

6. “Routes found are not guaranteed to be optimal. The probability of finding good paths increases with population size": This is also true for our case since it is built on the original LOST. However, robots will converge to the route that is most efficient among all discovered routes.

4.2.4 Trail-laying algorithm

Each robot has its local trail that is initially empty, and fresh crumbs will be added to them as robots experience events or receive trails on the network from other robots. In the original implementation, the received trail is transformed into the local coordinate system of the receiver. However, in our method, the received trail is combined with the robot's trail unchanged. While a robot is moving in an environment, it adds Crumbs to a temporary trail every S seconds; each Crumb is created using the current information of the path. Upon experiencing an event, the temporary trail will be shared with other robots and cleared afterwards. In the case of a large temporary trail, the robot will be re-spawned from the initial location with an empty temporary trail.

4.2.5 Trail-following algorithm

Robots seek the crumbs in the trail that take them closer to the goal event. This goal event can be finding a visual target in an environment or driving back home. For this reason, a robot needs to find a crumb that leads toward the goal and is reachable by the robot.

LOST: Finding the target crumb

Based on the properties of Crumbs, we can find the closest crumb to the target that is in the vicinity of the robot. Given the current location of the robot, L_r and the event E_g, crumbs in the trail can be filtered out based on their events and locations. The first filter is to check if the crumb's corresponding event is equal to the robot's target event: $E_g == E_{C_i}$, and the second filter is the reachability, crumbs whose location lies within a radius of the robot's current location: $dist(L_{C_i}, L_r) < r$, which in the metric space distance function is the Euclidean distance. After finding a set of reachable crumbs, the one with the minimum distance to the target will be chosen to be the target crumb. In the case of empty reachable crumbs, the robot will wander in the environment to either experience an event or find a reachable crumb to follow. As illustrated in Figure 4.1, the closest crumb to the target within the robot's sense radius is 7. The choice of r is important because, in a metric localization-space, a large radius causes the robot to ignore obstacles while a small radius will limit the robot's vicinity and causes slower convergence. A proper choice of sensing

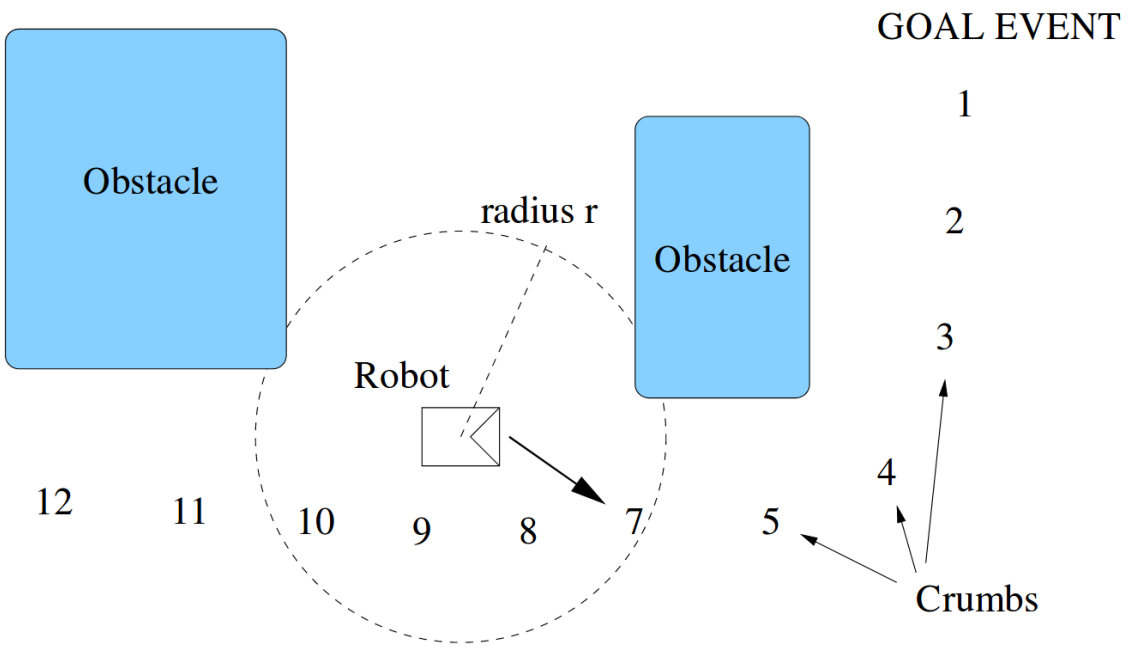


Figure 4.1: "The Trail-reading algorithm; a robot moves towards the Crumb within its sense radius r that has the lowest estimate of distance-to-goal." Vaughan et al. [96] (Copyright ©2002, IEEE)

radius r is a value larger than the distance between dropped crumbs and preferably lower than the size of the robot in the metric domain.

LOST: Moving toward the target crumb

Based on the target crumb, the robot can calculate the angle θ to the crumb and steer to that direction. Consequently, the robot moves towards the goal event. Since the shared trail is built by the entire population over time, robots benefit from each others' exploration. Each robot takes the shortest-discovered path to the goal from its location such that eventually, all the robots converge to the shortest discovered path from the home location to the goal location. The larger the population, the higher the probability of finding an efficient path.

Visual LOST: Finding the target crumb

In our proposed visual localization-space method, the overall algorithm is very similar. Crumbs are filtered out based on some constraints, crumbs' corresponding event should be equal to the robot's goal event, and the target crumb should be in the vicinity of the robot. In the metric space, vicinity can be checked by calculating the Euclidean distance from the robot to the crumbs. In the visual space, we can measure the distance between the crumbs by calculating cosine similarity between their embeddings and filtering out the ones whose

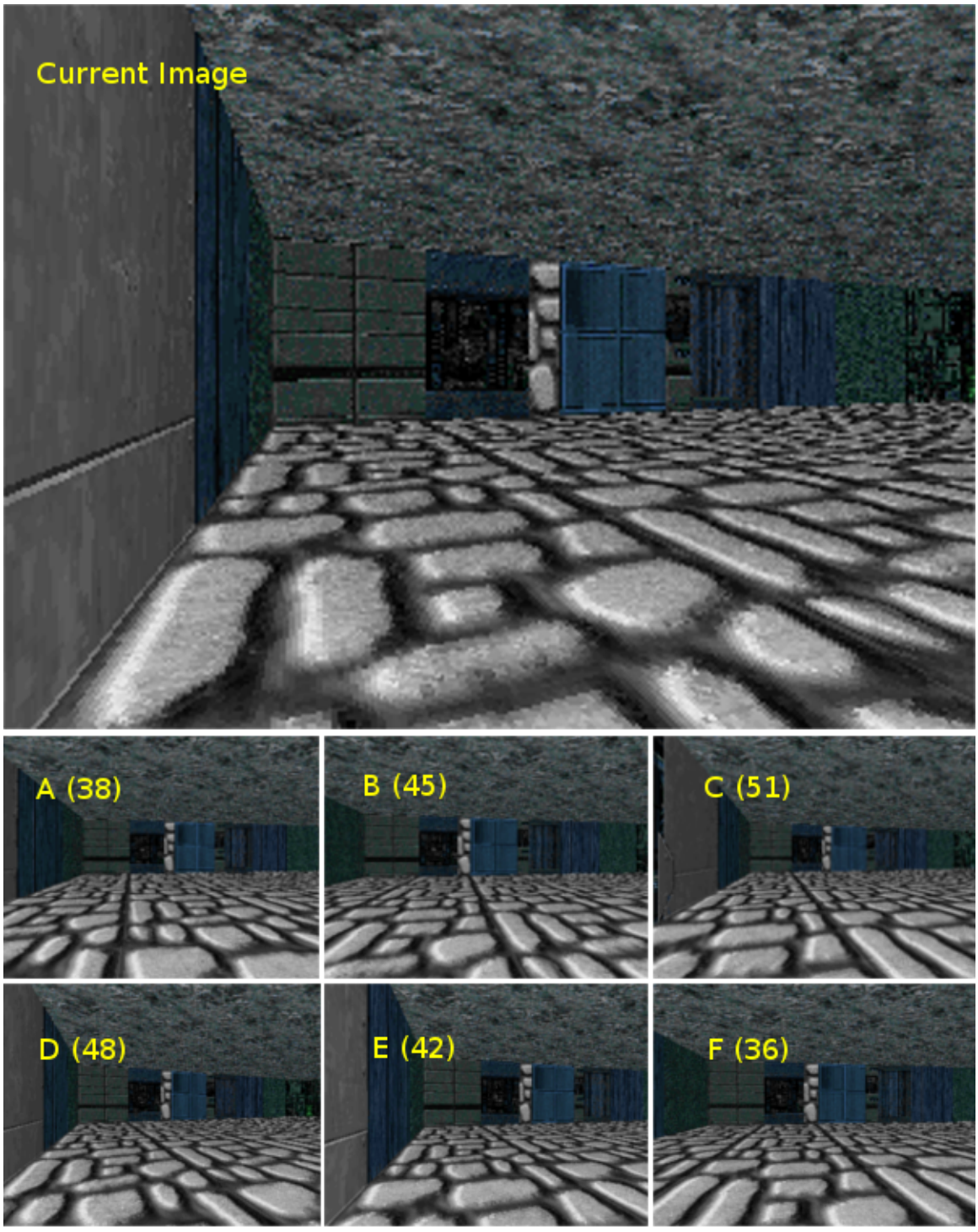


Figure 4.2: Current scene and crumbs in its vicinity extracted by the method, the numbers indicate estimates of distance-to-goal which in this case means Crumb F is the closest one to the goal (36 steps).

similarities are lower than a threshold. As shown in Figure 4.2, a set of crumbs that are in

the robot's vicinity are being selected, and among them, the one with the lowest estimate-to-goal is selected as the target crumb. The choice of r in metric space is important because of the obstacles. In visual space, a crumb behind an obstacle will not match the current image, however choosing a proper threshold is important for the fact that the target crumb should be reachable by LocoNet. A low threshold can cause selection of crumbs that cannot be reached by LocoNet, and a high threshold can cause selection of crumbs that are very close to the current scene, causing the robot to get stuck in a single location.

Although this algorithm for filtering out crumbs based on similarity and choosing the one that is closest to the goal event seems like a perfect solution, in visual localization-space, there are some issues such as perceptual-aliasing that makes it difficult for a robot to repeat the trail of crumbs towards the goal. The main problem is derived from the fact that there is no sense of metric distance between the current location and the crumb's location. There may be another scene taken from a different position in the environment that looks the same as the one the robot is seeing now. As discussed in the previous chapter, [75] proposed using K-nearest neighbour algorithm on a graph of waypoints to overcome some of these problems. K-nearest-neighbor automatically calculates the proper r to have the most accurate relocalization over all waypoints. However, in our case, we are not looking for the most similar crumb. We want to have a set of similar crumbs that have correctly been relocalized and based on them we choose the closest crumb to the goal. Building on [75], below we discuss our approach for finding the proper crumb that is reachable by LocoNet and presents an effective solution for the perceptual-aliasing problem.

First, instead of searching through all crumbs in the shared trail, we check all the local trails (each discovered by an agent) separately. We look up the N closest neighbors (most similar crumbs) in each local-trail (we chose $N = 10$ in our experiments). Nearest neighbours where the similarity score is lower than the lowest acceptable threshold (we chose 0.95 in our experiments) will be rejected, and if they are all lower than the threshold, no crumb will be selected as the target crumb. These selected crumbs are the most similar ones in the shared trail, and if we choose the closest crumb to the target, it might be very close to the current image and impossible to make progress towards the goal. For this reason, we have to find farther crumbs that are still reachable by the LocoNet but not so close to the current scene. Our solution to this problem is to look ahead in each local-trail sequentially and check if following crumbs have a good similarity to the current scene. For each neighbour, we start from the next crumb in the corresponding local-trail up to $L = 15$ of the next crumbs and add them to the reachable crumbs set if the similarity score is above a threshold ($\theta_{ahead} = 0.92$). The look-ahead process for each neighbour terminates when an upcoming crumb in its local-trail has similarity lower than the threshold. After generating the set of reachable crumbs, the one with the minimum distance to the goal event is selected to be the target crumb.

Although calculating the similarity between two embedding is fast (cosine similarity on a vector of 1024 numbers), to speed up this process and to account for the temporality factor, we added a step before running the nearest neighbour algorithm. Instead of calculating the similarities between the current image and all crumbs in the trail, we first run the procedure for the previously matched crumbs and their adjacent crumbs. If the resulting N nearest neighbours do not have similarity higher than a threshold, the procedure is called again for all crumbs.

Visual LOST: Moving toward the target crumb

In the previous chapter, we proposed a deep convolutional neural network to predict the proper action for the visual servoing task (given current and target images). The same network architecture is used here as well. We use a discretized action space: [Forward, Turn Right, Turn Left, Backward, Move Right, Move Left] and ResNet-18 [33] as a LocoNet architecture with an input size of $224 \times 224 \times 6$. Current and target images are stacked for the input to the network. The network generates action probabilities using softmax, and action is selected by sampling from the distribution.

This network is trained in a self-supervised manner. A dataset can be collected for this task, by randomly moving an agent in an environment and storing images before and after taking that action. As the network outputs six discrete classes, cross-entropy is used as the loss function.

4.3 Experiments

The following tests were performed to analyze the proposed localization-space and test the effectiveness of it in the task of finding a visual target in an unknown environment. We evaluate the system in a simulator, below we discuss our chosen testing environment.

4.3.1 Simulator

To evaluate our proposed method, we need to have a scalable simulator that is capable of having multiple agents equipped with cameras. During the development of this project, AirSim was unable to support multiple agents and performed slowly in a system equipped with an NVidia 1080 Ti. VizDoom [36] is a framework enabling developers to command *Doom* game agents in a 3D environment. Figure 4.3 shows an example of an image taken by an agent. Savinov et al in [75] provided textures and contents for this environment and implemented a tool to generate maps.

4.3.2 Task

All robots spawn from a specified initial location, and upon approaching the target location that we have selected for each map, receive a goal event. We only tested the system with one

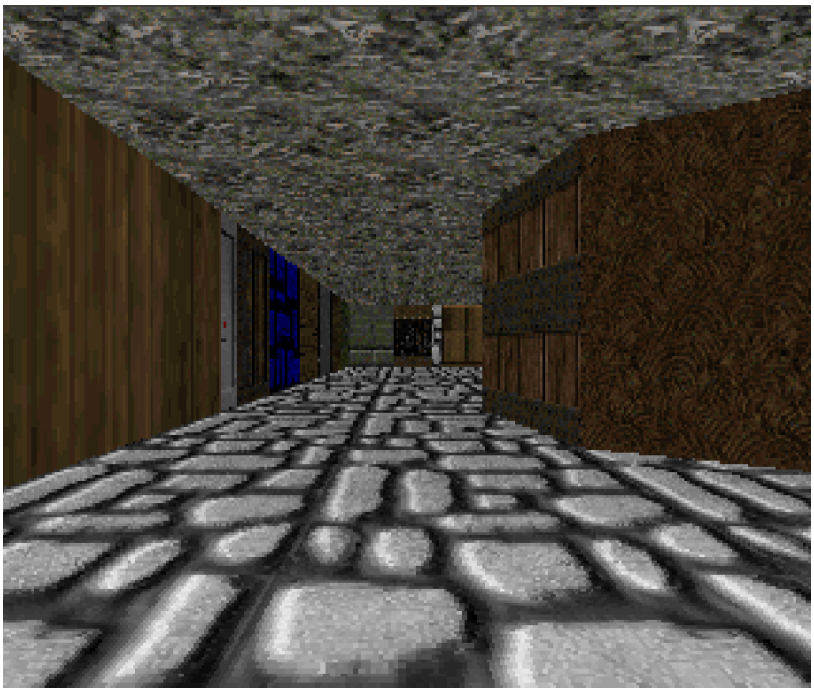


Figure 4.3: Example of an agent's viewpoint in VizDoom [36]. VizDoom provides APIs by which we can operate the agent. It also has tools for designing an environment and styling it.

event, but it is extendable to N events (such as visiting multiple places in each traverse) since each crumb has only one corresponding event and is separated from other events. Agents are equipped with an RGB-D camera providing colour and depth images. Depth image were only used for obstacle detection and calculating the distance to the left and right of the robot. There are 6 actions that an agent can take: Moving Forward, Backward, Left, Right and Turning Left and Right.

4.3.3 Training networks

The training procedure for both models is similar to the previous chapter. However, we have employed ResNet [33] architecture for both models. The Adam optimizer [39] is used to train both networks with learning rate = 0.001. Training data was generated online by running a wanderer agent in the training map. The training map is similar to the one [75] designed; it is a large puzzle-like environment with many routes and corridors. It has 400 variation of random textures on the floor and walls. These variations make it unlikely for the networks to over-fit. Testing maps are different structurally and in the type of textures used. At each training iteration for LocoNet, a mini-batch of $N = 32$ observations (current and target images) from replay memory buffer are fed to network with the truth label of the action that was taken at that moment. A similar procedure for the PlaceNet is used, but here each observation is made of three parts: an "anchor" which is the current image, a "positive

image" (that is an image temporally close to the anchor), and a “negative image" (that is an image taken from a different scene and temporally far from the anchor). Triplet loss is calculated by finding the cosine similarity between the anchor and the positive image, and the anchor and the negative image. We have discussed triplet loss and the training procedure in details in the previous chapter. The total number of images captured by the wanderer agent was 330,000. We split the data randomly into three sections: 300,000 images (90%) for training, 6,000 (2%) for validation, and 24,000 (8%) for testing. PlaceNet achieved an overall accuracy of 84.1% on testing data. Table 4.2 shows LocoNet performance on testing data, showing an overall accuracy of 73.4%.

Mode	Forward	Backward	Turn Right	Turn Left	Move Left	Move Right
Training	50,000	50,000	50,000	50,000	50,000	50,000
Validation	1,000	1,000	1,000	1,000	1,000	1,000
Testing	4,000	4,000	4,000	4,000	4,000	4,000

Table 4.1: Number of class data in the training, validation, and testing modes.

Class label	Precision	Recall	F1-score	Support
Forward	0.75	0.68	0.69	4000
Turn Right	0.69	0.75	0.69	4000
Turn Left	0.70	0.81	0.79	4000
Backward	0.77	0.68	0.70	4000
Move Right	0.72	0.71	0.68	4000
Move Left	0.73	0.67	0.67	4000
avg/total	0.73	0.71	0.70	24000

Table 4.2: Precision, recall and F1 score for LocoNet on test data.

4.3.4 Behavior

The behaviour module is in charge of navigating the robot in the environment. It considers obstacles, crumbs, and has a search strategy to explore the map. At each iteration, the robot estimates the distance to obstacles on the left and right of the robot.

Obstacle Avoidance

A robot initiates the obstacle avoidance behaviour when an obstacle is closer than a threshold $D = 30$ centimeters. Obstacles are detected using the depth channel of the image by simply finding the smallest depth pixel and then finding whether it is to the right of left side of the robot, or in the center. It then turns to the opposite direction from where the obstacle is located relative to the robot. Otherwise, the robot finds the closest-reachable crumb to the target by the algorithm described in section 4.2.5.

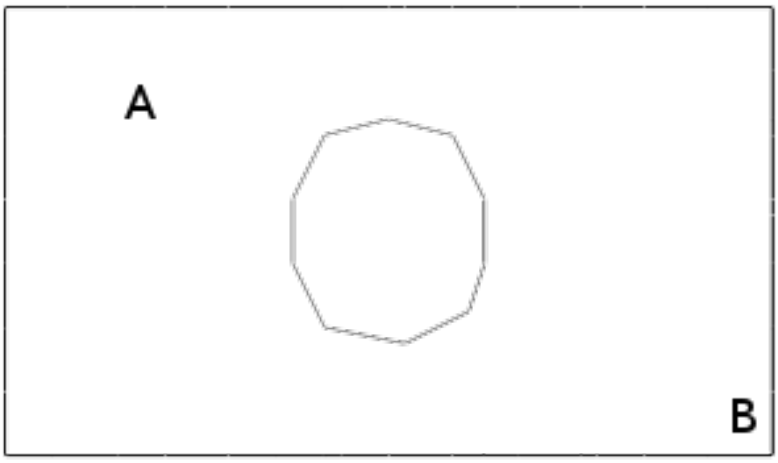


Figure 4.4: The top-down view of the experiment map structure.

Trail-Following Behavior

The target crumb image and the current image are stacked and given to the LocoNet to estimate the action taking the robot to the target crumb. LocoNet provides action probabilities; the final action is chosen by sampling from the action probabilities. In case no crumb was found as a target, the robot does the search behaviour outlined below.

Global Search Behavior

The simplest global search behaviour is to choose a random action among all the possible actions. As mentioned previously, the motion controller is only able to take six discrete actions, each in opposite directions. So taking random actions iteratively results in making no progress as all actions cancel out each others' movement (For example: Moving left and then Moving right). Our approach to search behaviour that makes progress is to have a queue of actions that are progressive and not against each other. We first randomly choose an action from a list: [Move Left, Move Right, Turn Left, Turn Right]. The randomly selected action is enqueued into the action queue N times. N is a random number between 2 and 7. Then the move forward action is enqueued to the action queue M times. M is also a random number ranged 2 to 7. This way we are sure that the robot goes forward and makes progress, as well as making turns and sideways moves. At each iteration of the global search behaviour, an action is dequeued from the action queue and is executed at that moment. The random action generator function is called when the queue is empty.

4.3.5 Experiment 1: Shortest path convergence

Setup

This experiment was conducted to verify that our method with visual localization-space trails causes agents to converge to the shortest discovered path. The top-down view of the

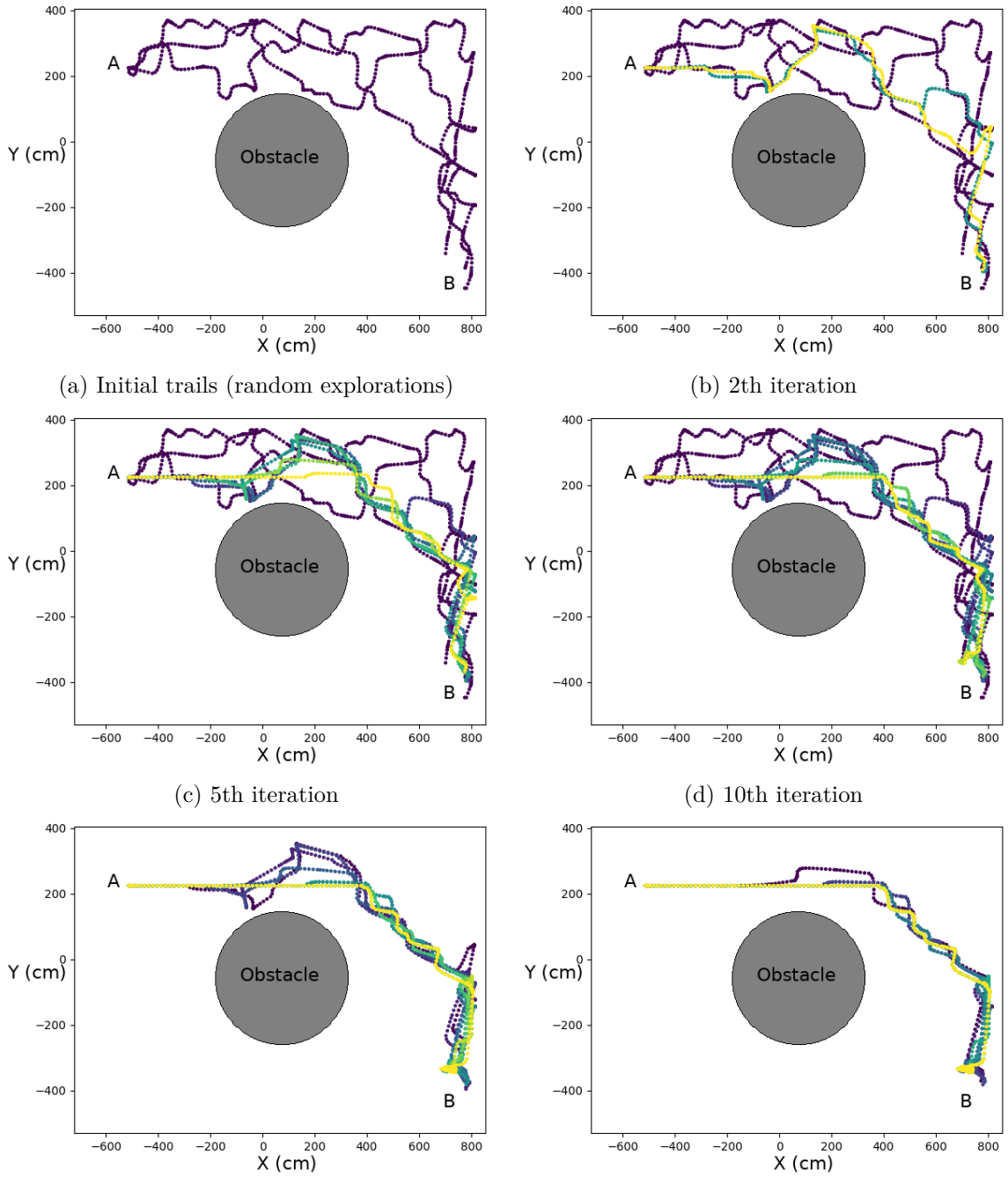
map is shown in Figure 4.4. Three agents start at the initial location (Place A). They have different arbitrary initial orientations. They begin to explore the environment and look for the target location (Place B), all with the same procedure discussed in the Behavior section. We have limited the size of the temporary trail to $L = 500$ to avoid having very long trails. Each robot respawns to the initial location upon finding the target place and waits for all other robots to find the target as well. We did this to make sure that there is more than one path to the target. As soon as all the robots have found a trail to the goal, they all start working again, and from then onwards they will not wait for other robots when they reach the goal.

Results

Figure 4.5 shows that after some iterations a robot gets stuck on the last crumb, trying to fit the current scene with the target crumb. The main reason for this is that there are no future crumbs from the last crumb to the goal and the look-ahead approach could not provide a better crumb, so that the target crumb would look the same as the current scene, making it difficult for the motion controller to select the right action. Given two similar images to the LocoNet, the output of the network (actions probability) will all be the same. In the original LOST paper, the authors had observed a similar problem and called it the *path-shortening* problem. Their proposed approach for it was to have a search strategy after reaching the final crumb and wander in that area until the robot receives the goal event. They verified the effectiveness of their solutions with separate experiments. In our case, there are some other possible approaches to this problem. Since the goal image is given, after reaching the final crumb, a visual servoing (by LocoNet) can be done given the current and goal images. However, to simplify our method, we have replicated the LOST's strategy by running the global search behaviour after reaching the last crumb.

We performed the original LOST with the same initial trails that each robot found. Global positions were used as the localization-space. Figure 4.6 shows the resulting trajectories and how it not only converged to the shortest discovered path but also it optimized that path over a few iterations. A higher number of agents results in having a higher chance to find the shortest path. The LOST trail-following method also makes the path smoother on sharp turns. Our proposed visual localization-space can converge to the shortest discovered path, but it fails at path smoothing. The main reason for this problem is that two scenes on a corner look different and PlaceNet is not able to look-ahead in the path and choose a crumb from another viewpoint. Even if it does, LocoNet is not able to drive the robot towards the selected target crumb. Although in metric localization-space two crumbs on a corner are within vicinity and reachability of each other, in the visual localization-space they are not.

One solution to this problem is to use a wider FOV camera (or two additional cameras on the sides), and have a sliding-window comparing crumbs at different views. As an example,



(a) Initial trails (random explorations)

(b) 2th iteration

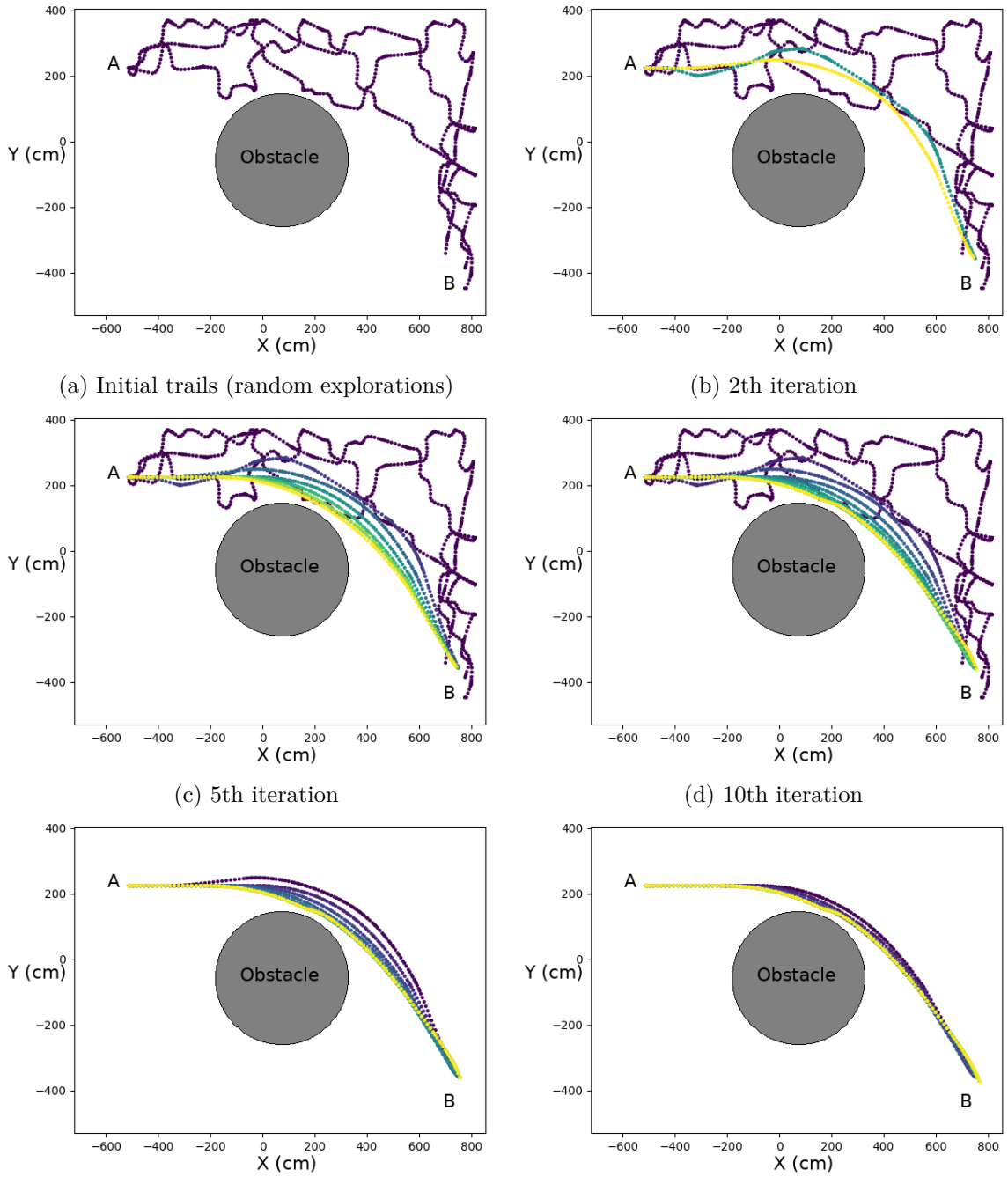
(c) 5th iteration

(d) 10th iteration

(e) 12th iteration

(f) 15th iteration

Figure 4.5: Experiment 1: the robots' trajectories over time (three robots). From top-left the initial trails discovered by each robot to bottom-right the 15th iteration where robots have converged on the shortest-discovered path. Colors indicates trails age; yellow is the most recent trail to purple the oldest one that is almost evaporated. Path-shortening problem happened at the end of the trail where an unexpected number of crumbs are crowded.



(a) Initial trails (random explorations)

(b) 2th iteration

(c) 5th iteration

(d) 10th iteration

(e) 12th iteration

(f) 15th iteration

Figure 4.6: Experiment 1: robots' trajectories running original LOST implementation. From top-left the initial trails discovered by each robot to bottom-right the 15th iteration where robots have converged on the shortest path (smoothing all the sharp turns in the trail).

on turning at a right-hand corner, the image taken from the right camera before the corner would look like the same as the frontal view of the robot after the corner (since they

may have the same viewpoint). Unfortunately, VizDoom does not support having multiple cameras or a wider camera, so we kept to the default camera settings and specifications.

Another solution regarding the sharp corners' vicinity problem is to have a local-search behaviour that would do a local optimization. This approach may not solve this problem as fast and efficiently as metric localization-space would do, but over a long period, it can make many improvements. We modified the behaviour module with the following approach:

Considering an ϵ-greedy algorithm, we added a factor ϵ to the trail-following behaviour. At each iteration, a robot follows the target crumb with a probability of $\epsilon = 0.6$ or it does a short random movement (akin to global-search behaviour) otherwise. In the beginning, the value of ϵ is higher which makes robots do more local optimization, and it is decreased by another factor $C_\epsilon = 0.001$ at each iteration (As with ϵ-greedy policy in reinforcement learning). We have also added another step to stabilize the shortest discovered path and keep it from evaporating. All robots disable the local-search behaviour to do only the trail-following behaviour before the best-discovered path is evaporated which acts as a stabilizing factor in optimization.

4.3.6 Experiment 2: Trail-end Event Searching

Setup

Similar to LOST, we have conducted another experiment to evaluate our solution to the trail-shortening problem proposed in the previous experiment. The experiment setup and procedures are the same as the previous experiment. The shared trail is initialized first with local trails explored by robots. As discussed earlier, the path-shortening problem happens when a robot gets stuck in the last crumb tying to fit the current and target scenes, and by chance, it goes within range of the goal and receives the goal event. We decided to run the global search behaviour after reaching the last crumb in the trail (the crumb having 1 step to the target).

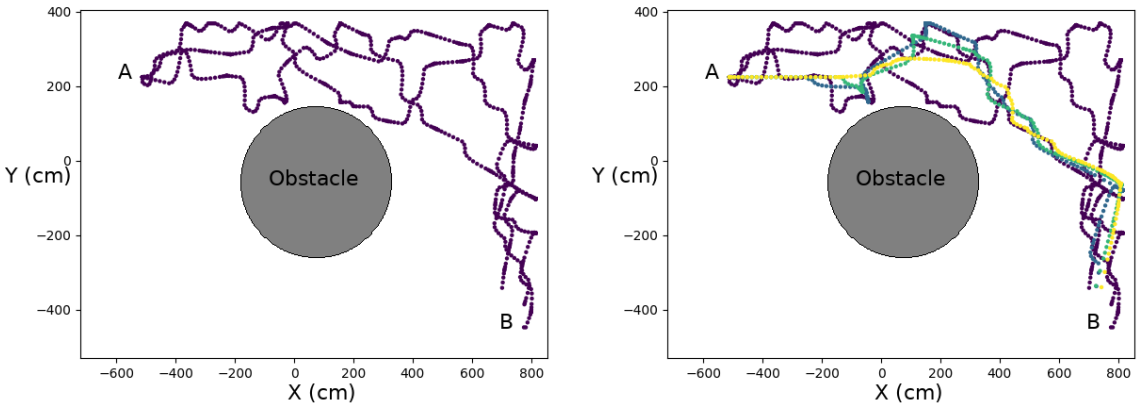
Results

Figure 4.7 shows the robots' trajectories from the initial location (A) to the goal location (B) over time (40 iterations) and how the added extra step to the approach solved the path-shortening problem. As soon as a robot reaches the last crumb, it does a global search behaviour which is mostly going forward and avoiding obstacles.

4.3.7 Experiment 3: Local optimization

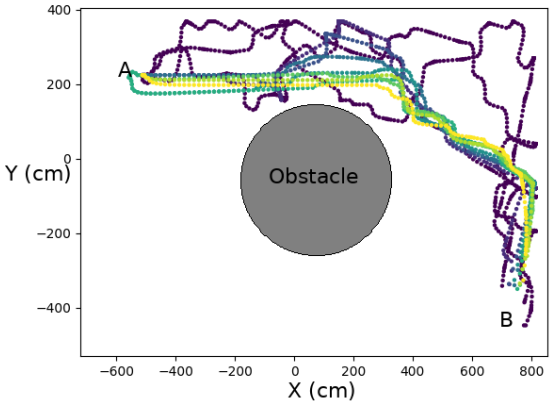
Setup

This experiment was performed to evaluate the effectiveness of the local-search solution to the sharp-edges' vicinity problem. The setup and initial trails are identical to the previous experiment. Filling the shared trail with initial trails explored by each robot from the

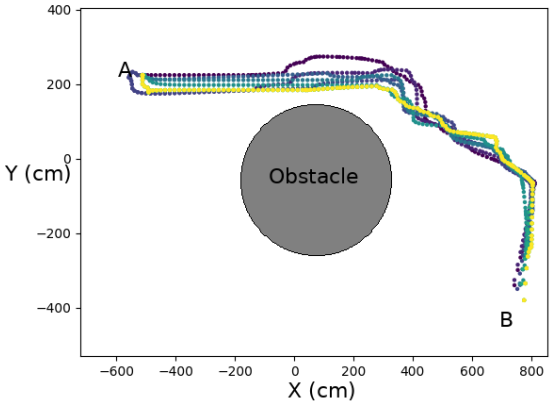


(a) Initial trails (random explorations)

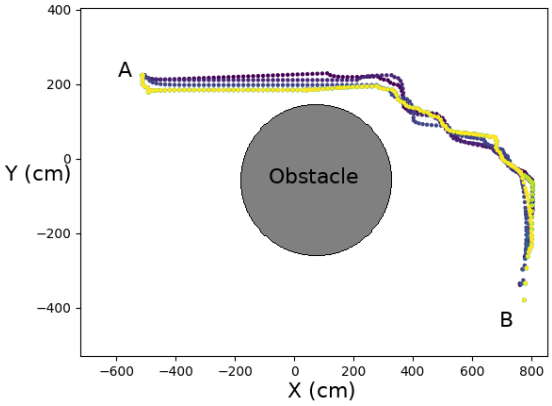
(b) 2th iteration



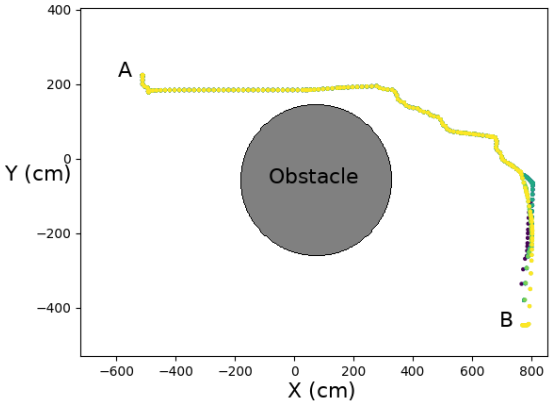
(c) 5th iteration



(d) 10th iteration



(e) 12th iteration

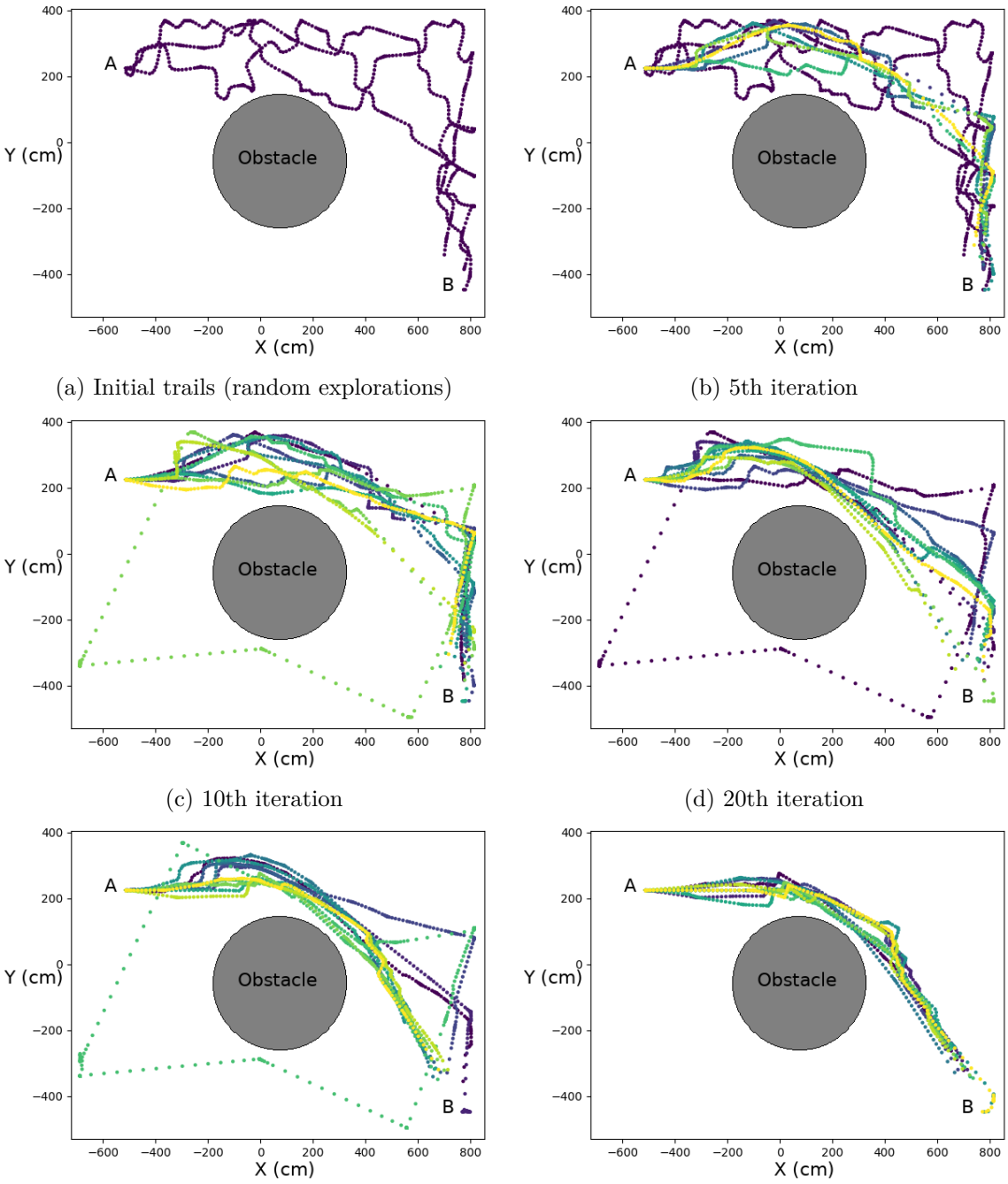


(f) 15th iteration

Figure 4.7: Experiment 2: the robots' trajectories. From top-left the initial trails discovered by each robot to bottom-right the 15th iteration where the robots have converged on the shortest-discovered path and managed path-shortening problem as well.

previous experiment lets us have a fair comparison with the results of the previous experiment. We have collected trajectories with 15 trials due to the randomness of the local-search controller.

Results



(a) Initial trails (random explorations)

(b) 5th iteration

(c) 10th iteration

(d) 20th iteration

(e) 30th iteration

(f) 35th iteration

Figure 4.8: Experiment 3: the robots' trajectories over time. From top-left the initial trails discovered by each robot to bottom-right the 35th iteration where robots have converged on the shortest path in (smoothing the sharp-turns). Diverged trails explored by local search behavior are seen in the middle row (20th and 30th iteration).

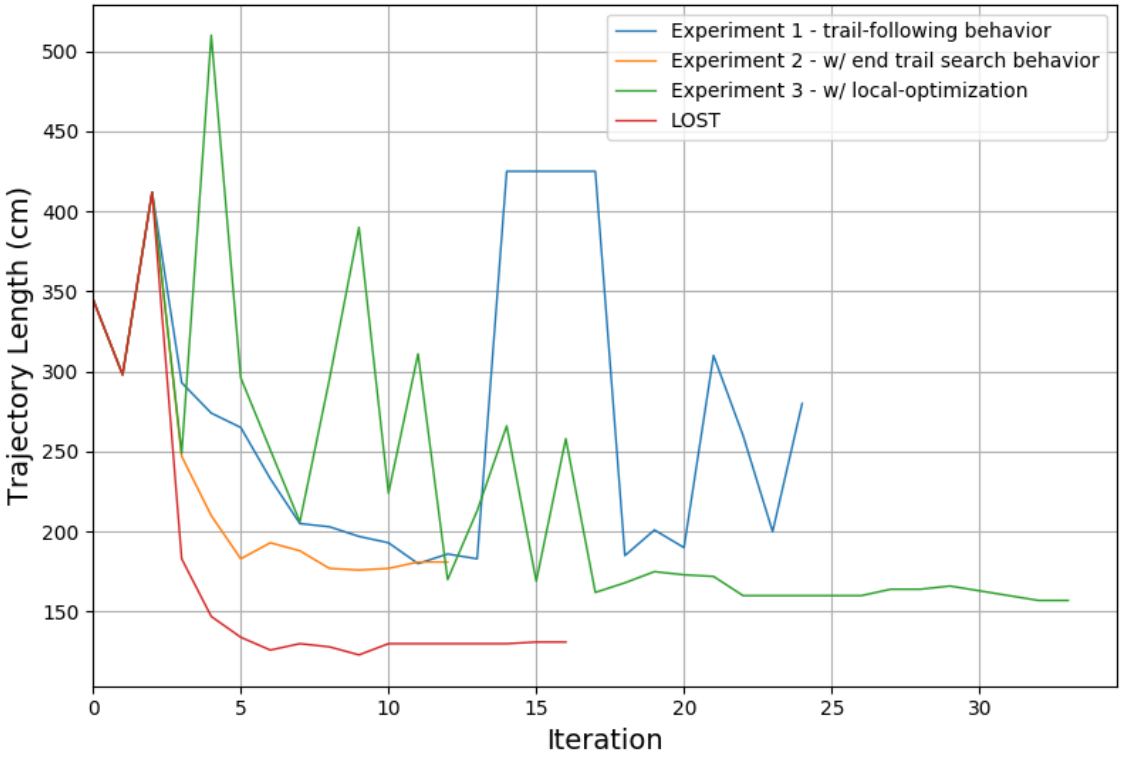


Figure 4.9: Trajectory length over number of iterations. First three iterations are initial trails that are set to be same.

Figure 4.8 shows the robots' trajectories over time from the initial trails to 35th iteration. Agents begin with a noticeable local search behaviour which results in diverging trajectories. They also lost the trail and did not reach the goal in some trials which is the nature of this optimization process. Eventually, their local-search behaviour slows down and stops. Figure 4.9 depicts all trajectories over time in each experiment. The original LOST implementation shows promising results on time to converge and final trajectory length. The blue line shows the first experiment performance, starting well for a few iterations and then failing to converge at around iteration 15, showing the path-shortening problem. The orange line shows the second experiment with end-trail search behaviour and how it converges quickly on the shortest discovered path (with some extra optimization). The green line shows the experiment 3 implementation with local optimization, from large trajectory lengths that gradually shorten resulting in a shorter path compared to the other two implementations. It takes three times longer for agents to converge compared to the second experiment (no local-search behaviour). However, there is a higher chance that this method can find a shorter and more efficient path to the target.

A histogram of trajectory lengths is shown in Figure 4.10 with fitted Gaussians. We ran each experiment 10 times and recorded the final trajectory length at the 45th iteration. The original LOST implementation and the second experiment with end-trail search behaviour

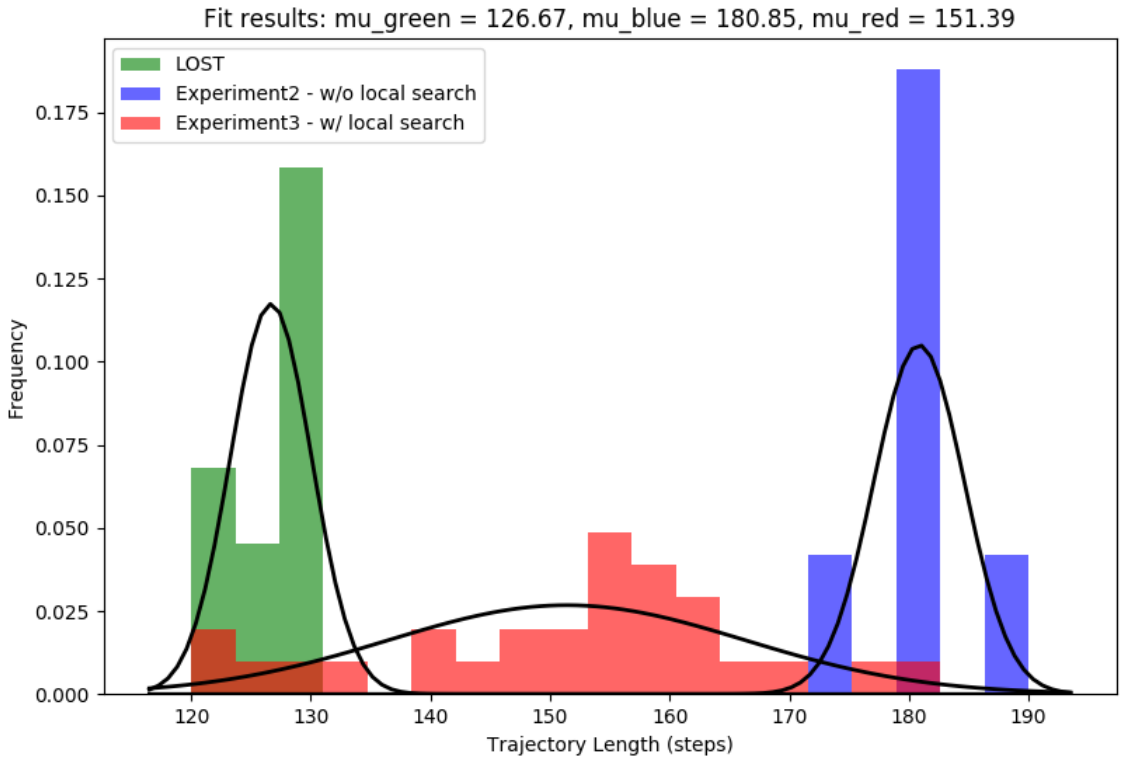


Figure 4.10: Repeat completion times for identical environment (green), light-changed environment (red) and object-moved environment (blue).

have an average of 15 iterations to converge, but for the local-search version, this number is 45, which is when the local-search behaviour stops. As the histogram shows, the LOST implementation and end-trail search behaviour have steady and less diverse trajectory lengths in different trials. However, the local-search behaviour performance ranges between the optimal solution to the average results of end-trail search behaviour. The reason for having the upper limit being close to end-trail search behaviour is the stabilization factor which does not let the system forget the best-discovered path by disabling the local-search behaviour once the best-discovered path is nearly evaporated.

Although the visual localization-space with local-search approach does not converge as fast as the metric localization-space, it has many advantages over the metric space; visual localization-space is reference-free and has independent scents that are drift-free, i.e. cause zero cumulative error. Apart from that, local-search behaviour is not practical in real-world applications since there is only a little chance of improvement on each trial a robot does. However, a sliding window approach with a wider FOV camera or multiple cameras having different viewpoint should theoretically work as efficiently as metric localization-space.

4.4 Conclusion, limitations and future work

In this chapter, we introduce a new localization-space for robot teams allowing them to share reference-free landmarks of an environment for the navigational task. In the previous chapter, we show that deep models are capable of extracting crucial features in a scene that are condition- and viewpoint-invariant and can be used for relocalization and visual-servoing tasks. We demonstrate that with well-trained deep models for place recognition and visual-servoing, a robot is able to repeat a taught long path successfully. We extend this method and apply it to Localization-space Trails for Robot Teams (LOST) [96] as a new localization-space instead of the original metric-space. We redefine some of the concepts and evaluation constraints the authors proposed in the original paper; one of the main constraints in LOST is the ability to repeat a path that a robot has already gone through which we show in the previous chapter.

Employing visual landmarks as the localization-space has many advantages: almost all metric localization-spaces are reference-dependent; there should be a position and orientation chosen as a reference point, and all positions are relative to that reference point. For the Global Positioning System (GPS) this is not a problem since the reference point is static and fairly accurate, but GPS is not accessible in all environments. For almost all other cases there is either the problem of setting up sensors in an environment (which for a navigation task in an unknown environment is not possible) or the drifting localization problem (cumulative error caused by small errors which results in a huge error over time). LOST was proposed to fix the drifting localization problem and to manage robots to have a shared localization-space, but as the authors stated: “Localization must be good enough to enable the robot to make a single traverse." [96]. Visual landmarks are reference-independent and self-descriptive; a robot can relocalize itself in an environment given a set of landmarks without a reference point. It can also be used by aerial robots where all sensors are noisy and maintaining accurate odometry is hard.

This task could have also been done by reinforcement learning using policy gradient [101, 65, 77] in an end-to-end fashion. In a policy gradient approach, a model is enforced to learn a policy that maximizes its compatibility with an environment, or in other words, a reward function. Considering shortest-path convergence problem only, apart from multi-agent navigation, a reward function can be defined for an agent to converge to the shortest path: 1. a positive reward for each action that results in the robot being closer to the target 2. a negative reward (punishment) that results in the robot being further from the target 3. a very high negative reward for colliding with an obstacle. 4. a very high positive reward for reaching the goal and 5. inverted total travelling time. However, such a strategy to train a model that only works for one path in one environment is not practical. A typical reinforcement learning procedure requires many more training iterations to figure out a

relation between the given reward and the raw input. In contrast, our approach requires only a few number of iterations to converge.

We test our method in the VizDoom simulator. VizDoom is an interface by which we can control Doom agents through APIs. We have trained our deep models (place recognition and visual-servoing) in complex environments with random textures. The method is then tested by three robots in a different environment with a different set of textures for the task of finding a visual goal. As illustrated in our experiment section, in all the cases the robots converge on the shortest-discovered path. We also add extra behavior steps for path-shortening and sharp edges (low angle-of-view) problems. The end-event search behaviour eliminates the path-shortening problem. However, our local-search behaviour for sharp-edges was not as efficient as other methods. This was expected since it has an extra optimization step that works like ϵ-greedy and takes longer to converge.

As with the previous chapter, our method depends highly on the deep models and consequently our training data and procedure. A diverse and comprehensive dataset makes deep models able to generalize better and extract highly informative features such as textures and salient objects. Training and testing on real-world data could suggest directions for future work. As deep models are black-boxes that are trainable for a different objectives, we may need to test them under many different circumstances.

The main drawback of our method is the size of the crumbs. We planned to share a 1024-size vector of deep-features only, but for the simplicity of training and network architecture, we chose to have two separate networks for place recognition and visual-servoing. Because of that, crumbs must contain the actual image of a scene (needed by the visual servoing module) as well as the feature vector from PlaceNet (needed by the relocalization module). Having one network able to do both tasks of place recognition and visual-servoing is an important feature that could make this method practical. This can be done by training a model with a Siamese architecture [40], having parallel pipelines for two images followed by a fully connected network that inputs the representations and outputs actions labels or “Not reachable" label. Based on the output class label and their probability, we can estimate their similarity and decide on a behaviour accordingly.

Applying a sliding-window approach with a wider FOV camera to overcome the sharp-edges’ scene changing problem, and a smart initial search behaviour which takes into account other agents’ explorations are other possible future work.

Chapter 5

Conclusion

In this thesis, we have explored the use of visual landmarks in robot navigation with the aid of deep convolutional neural networks. Previous approaches in navigation employ either classical computer vision techniques to extract visual feature for appearance-based localization or other sensory information that suffer from calibration and drift. Our proposed deep visual landmarks are robust to different conditions and environmental changes. Three applications were demonstrated in this regards: 1) UAV Visual Teach and Repeat using only semantic object features. 2) Visual Teach and Repeat using deep CNN features generated in an end-to-end manner. 3) Visual localization-space trails for multi-agent navigation.

The work presented in Chapter 2 explored the use of semantic object features provided by a deep-CNN object detector for visual teach and repeat. We demonstrated the method with real UAV experiments, but in an lab/office-like environment as a substitute for our eventual goal of large scale outdoor flights. Semantic objects are detectable in different viewpoint and lighting condition with interesting novel invariances to the object being completely turned around or replaced by another of the same category. The proposed method can repeat the path completely with a start location 5m away from the taught initial position, is invariant for up to 40% changes in altitude, as well as environment and lighting changes.

In Chapter 3 we demonstrated a visual teach and repeat system that uses deep convolutional neural networks for localization and navigation. Two separate models were used in this system: 1) the PlaceNet model generates feature vectors by which two images can be compared in terms of similarity. 2) the LocoNet model predicts actions for short-range visual navigation. Deep CNN models are popular due to their ability to extract high-level features automatically with no need for an expert. Both models were trained on a dataset collected from a simulator. We evaluated the system in the simulator. The system showed good performance in different scenarios except in the lighting variation test. The primary cause of it was that both models were not trained in different lighting conditions (due to the lack of such an API in AirSim).

In Chapter 4 we presented a cooperative navigation system that uses deep visual landmarks as a shared map among all robots. This system is built upon Localization-Space

Trails for Robot Teams (LOST) [96], and employs the deep visual teach and repeat system from Chapter 3 for visual feature extraction and short-range navigation. Visual features are used as a new localization-space over the typical metric-space that is used in LOST for localization. We have evaluated the system in a simulator to proof the concept; both models were trained based on data collected from the simulator. We demonstrated that the proposed localization-space is sufficient for cooperative navigation. We have also proposed and evaluated techniques that help the system converge on smoother paths. Beside all the advantages the proposed visual localization-space has over metric-space, designing and training the deep models to perform in real-world scenarios is challenging.

We believe that visual information may be more practical in real-world scenarios and have many advantages over metric-space sensors including reference-dependence and drift. However, building a navigation-localization system that only uses visual information has its challenges. Although deep neural networks have resolved issues such as feature-extraction, they need to be carefully designed and trained with a comprehensive dataset to make them broadly useful and invariant to different conditions and environments. With the ongoing attention towards the deep neural networks and their usages in robotics and navigation, reaching this goal is not too far and out of reach.

Bibliography

[1] Alaa Eldin Abdelaal, Maram Sakr, and Richard Vaughan. Lost highway: a multiple-lane ant-trail algorithm to reduce congestion in large-population multi-robot systems. In *Canadian Conference on Computer and Robot Vision (CRV)*, pages 161–167. IEEE, 2017.

[2] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[3] Shehroze Bhatti, Alban Desmaison, Ondrej Miksik, Nantas Nardelli, N. Siddharth, and Philip H. S. Torr. Playing doom with slam-augmented deep reinforcement learning. *CoRR*, abs/1612.00380, 2016.

[4] Francisco Bonin-Font, Alberto Ortiz, and Gabriel Oliver. Visual navigation for mobile robots: A survey. *Journal of Intelligent and Robotic Systems*, 53(3):263–296, 2008.

[5] Sean L Bowman, Nikolay Atanasov, Kostas Daniilidis, and George J Pappas. Probabilistic data association for semantic slam. In *International Conference on Robotics and Automation (ICRA)*, pages 1722–1729. IEEE, 2017.

[6] Jake Bruce, Niko Sünderhauf, Piotr Mirowski, Raia Hadsell, and Michael Milford. One-shot reinforcement learning for robot navigation with interactive replay. *CoRR*, abs/1711.10137, 2017.

[7] Jake Bruce, Jens Wawerla, and Richard Vaughan. The sfu mountain dataset: Semi-structured woodland trails under changing environmental conditions. *International Conference on Robotics and Automation (ICRA) WS VPRiCE*, 2015.

[8] BA Cartwright and TS Collett. How honey bees use landmarks to guide their return to a food source. *Nature*, 295(5850):560, 1982.

[9] Zetao Chen, Adam Jacobson, Niko Sünderhauf, Ben Upcroft, Lingqiao Liu, Chunhua Shen, Ian Reid, and Michael Milford. Deep learning features at scale for visual place recognition. In *International Conference on Robotics and Automation (ICRA)*, pages 3223–3230. IEEE, 2017.

[10] Zetao Chen, Obadiah Lam, Adam Jacobson, and Michael Milford. Convolutional neural network-based place recognition. *CoRR*, abs/1411.1509, 2014.

[11] Zhichao Chen and Stanley T. Birchfield. Qualitative vision-based path following. *IEEE Transactions on Robotics*, 25(3):749–754, 2009.

[12] Lee Clement, Jonathan Kelly, and Timothy D Barfoot. Robust monocular visual teach and repeat aided by local ground planarity and color-constant imagery. *Journal of Field Robotics*, 34(1):74–97, 2017.

[13] Mark Cummins and Paul Newman. Fab-map: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research*, 27(6):647–665, 2008.

[14] Vikas Dhiman, Shurjo Banerjee, Brent Griffin, Jeffrey Mark Siskind, and Jason J. Corso. A critical investigation of deep reinforcement learning for navigation. *CoRR*, abs/1802.02274, 2018.

[15] Dokyo. Modular neighborhood pack. `https://www.unrealengine.com/marketplace/modular-neighborhood-pack`, 2016. Accessed: 2017-12-17.

[16] Marco Dorigo. Optimization, learning and natural algorithms. *PhD Thesis, Politecnico di Milano*, 1992.

[17] Marco Dorigo and Gianni Di Caro. Ant colony optimization: a new meta-heuristic. In *Proceedings of The 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 2, pages 1470–1477. IEEE, 1999.

[18] Marco Dorigo and Luca Maria Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[19] Marco Dorigo, Vittorio Maniezzo, Alberto Colorni, et al. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 26(1):29–41, 1996.

[20] Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2017.

[21] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *Robotics and Automation Magazine*, 13(2):99–110, 2006.

[22] F Dyer. Spatial memory and navigation by honeybees on the scale of the foraging range. *Journal of Experimental Biology*, 199(1):147–154, 1996.

[23] Fred C Dyer and Thomas D Seeley. Dance dialects and foraging range in three asian honey bee species. *Behavioral Ecology and Sociobiology*, 28(4):227–233, 1991.

[24] Felix Endres, Jürgen Hess, Nikolas Engelhard, Jürgen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the rgb-d slam system. In *International Conference on Robotics and Automation (ICRA)*, pages 1691–1696, 2012.

[25] Patrick Foo, William H Warren, Andrew Duchon, and Michael J Tarr. Do humans integrate routes into a cognitive map? map-versus landmark-based navigation of novel shortcuts. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 31(2):195, 2005.

[26] Matthias O Franz and Hanspeter A Mallot. Biomimetic robot navigation. *Robotics and Autonomous Systems*, 30(1-2):133–153, 2000.

[27] Ryusuke Fujisawa, Hikaru Imamura, Takashi Hashimoto, and Fumitoshi Matsuno. Communication using pheromone field for multiple robots. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 1391–1396. IEEE, 2008.

[28] Paul Furgale and Timothy D Barfoot. Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics*, 27(5):534–560, 2010.

[29] Sabine Gillner and Hanspeter A Mallot. Navigation and acquisition of spatial knowledge in a virtual maze. *Journal of Cognitive Neuroscience*, 10(4):445–463, 1998.

[30] Arren J Glover, William P Maddern, Michael J Milford, and Gordon F Wyeth. Fab-map+ ratslam: Appearance-based slam for multiple times of day. In *International Conference on Robotics and Automation (ICRA)*, pages 3507–3512. IEEE, 2010.

[31] Katalin M Gothard, William E Skaggs, Kevin M Moore, and Bruce L McNaughton. Binding of hippocampal ca1 neural activity to multiple reference frames in a landmark-based navigation task. *Journal of Neuroscience*, 16(2):823–835, 1996.

[32] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7272–7281. IEEE Computer Society, 2017.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[34] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research*, 31(5):647–663, 2012.

[35] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2017.

[36] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016.

[37] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *International Conference on Computer Vision (ICCV)*, pages 2938–2946, 2015.

[38] Arbaaz Khan, Vijay Kumar, and Alejandro Ribeiro. Learning sample-efficient target reaching for mobile robots. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 3080–3087. IEEE, 2018.

[39] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[40] Gregory Koch. Siamese neural networks for one-shot image recognition. In *International Conference on Machine Learning (ICML) Deep Learning Workshop*, 2015.

[41] Kurt Konolige, Motilal Agrawal, Robert C Bolles, Cregg Cowan, Martin Fischler, and Brian Gerkey. Outdoor mapping and navigation using stereo vision. In *Experimental Robotics*, pages 179–190. Springer, 2008.

[42] Tomáš Krajník, Pablo Cristóforis, Keerthy Kusumam, Peer Neubert, and Tom Duckett. Image features for visual teach-and-repeat navigation in changing environments. *Robotics and Autonomous Systems*, 88:127–141, 2017.

[43] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision (ECCV)*, pages 21–37. Springer, 2016.

[44] Stephanie Lowry, Niko Sünderhauf, Paul Newman, John J Leonard, David Cox, Peter Corke, and Michael J Milford. Visual place recognition: A survey. *IEEE Transactions on Robotics*, 32(1):1–19, 2016.

[45] Kirk MacTavish, Michael Paton, and Timothy D Barfoot. Visual triage: A bag-of-words experience selector for long-term visual route following. In *International Conference on Robotics and Automation (ICRA)*, pages 2065–2072. IEEE, 2017.

[46] Ralf Mayet, Jonathan Roberz, Thomas Schmickl, and Karl Crailsheim. Antbots: A feasible visual emulation of pheromone trails for swarm robots. In *International Conference on Swarm Intelligence*, pages 84–94. Springer, 2010.

[47] Michael J Milford and Gordon F Wyeth. Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights. In *International Conference on Robotics and Automation (ICRA)*, pages 1643–1649. IEEE, 2012.

[48] Michael J Milford, Gordon F Wyeth, and David Prasser. Ratslam: a hippocampal model for simultaneous localization and mapping. In *International Conference on Robotics and Automation (ICRA)*, volume 1, pages 403–408. IEEE, 2004.

[49] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andy Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2017.

[50] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 1928–1937, 2016.

[51] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[52] Ralf Möller, Dimitrios Lambrinos, Rolf Pfeifer, Thomas Labhart, and Rüdiger Wehner. Modeling ant navigation with an autonomous agent. *Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, 1998.

[53] Beipeng Mu, Shih-Yuan Liu, Liam Paull, John Leonard, and Jonathan P How. Slam with objects using a nonparametric pose graph. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 4602–4609. IEEE, 2016.

[54] Lynn Nadel and Lloyd MacDonald. Hippocampus: Cognitive map or working memory? *Behavioral and Neural Biology*, 29(3):405–409, 1980.

[55] Trung Nguyen, George KI Mann, Raymond G Gosine, and Andrew Vardy. Appearance-based visual-teach-and-repeat navigation technique for micro aerial vehicle. *Journal of Intelligent and Robotic Systems*, 84(1-4):217–240, 2016.

[56] Junhyuk Oh, Valliappa Chockalingam, Satinder P. Singh, and Honglak Lee. Control of memory, active perception, and action in minecraft. In *International Conference on Machine Learning (ICML)*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2790–2799. JMLR.org, 2016.

[57] Kazunori Ohno, Takashi Tsubouchi, Bunji Shigematsu, Shoichi Maeyama, and ShinâĂŹichi Yuta. Outdoor navigation of a mobile robot between buildings based on dgps and odometry data fusion. In *International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1978–1984. IEEE, 2003.

[58] John O'keefe and Lynn Nadel. *The hippocampus as a cognitive map*. Oxford: Clarendon Press, 1978.

[59] Stefano Panzieri, Federica Pascucci, and Giovanni Ulivi. An outdoor navigation system using gps and inertial platform. *IEEE/ASME Transactions on Mechatronics*, 7(2):134–142, 2002.

[60] Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2018.

[61] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *Advances in Neural Information Processing Systems (NIPS-W)*, 2017.

[62] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

[63] Michael Paton, Kirk MacTavish, Michael Warren, and Timothy D Barfoot. Bridging the appearance gap: Multi-experience localization for long-term visual teach and repeat. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 1918–1925. IEEE, 2016.

[64] David Payton, Mike Daily, Regina Estowski, Mike Howard, and Craig Lee. Pheromone robotics. *Autonomous Robots*, 11(3):319–324, 2001.

[65] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008.

[66] Andreas Pfrunder, Angela P Schoellig, and Timothy D Barfoot. A proof-of-concept demonstration of visual teach and repeat on a quadrocopter using an altitude sensor and a monocular camera. In *Canadian Conference on Computer and Robot Vision (CRV)*, pages 238–245. IEEE, 2014.

[67] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525. IEEE Computer Society, 2017.

[68] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 91–99, 2015.

[69] John G Rogers, Alexander JB Trevor, Carlos Nieto-Granda, and Henrik I Christensen. Simultaneous localization and mapping with learned object recognition and semantic data association. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 1264–1270. IEEE, 2011.

[70] Andrew Russell, David Thiel, and Alan Mackay-Sim. Sensing odour trails for mobile robot navigation. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 2672–2677. IEEE, 1994.

[71] R Andrew Russell. Ant trails-an example for robots to follow? In *International Conference on Robotics and Automation (ICRA)*, volume 4, pages 2698–2703. IEEE, 1999.

[72] Seyed Abbas Sadat and Richard T Vaughan. So-lost-an ant-trail algorithm for multi-robot navigation with active interference reduction. In *Conference on Artificial Life (ALIFE)*, pages 687–693, 2010.

[73] Seyed Abbas Sadat and Richard T Vaughan. Bravo: Biased reciprocal velocity obstacles break symmetry in dense robot populations. In *Canadian Conference on Computer and Robot Vision (CRV)*, pages 441–447. IEEE, 2012.

[74] Renato F Salas-Moreno, Richard A Newcombe, Hauke Strasdat, Paul HJ Kelly, and Andrew J Davison. Slam++: Simultaneous localisation and mapping at the level of objects. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1352–1359, 2013.

[75] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2018.

[76] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823, 2015.

[77] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[78] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, pages 621–635. Springer, 2018.

[79] Titus Sharpe and Barbara Webb. Simulated and situated models of chemical trail following in ants. In *Proceeding of the Fifth International Conference on Simulation of Adaptive Behavior*, pages 195–204, 1998.

[80] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

[81] Kwang Mong Sim and Weng Hong Sun. Ant colony optimization for routing and load-balancing: survey and new directions. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 33(5):560–572, 2003.

[82] Zhao Song, Seyed Abbas Sadat, and Richard T Vaughan. Mo-lost: Adaptive ant trail untangling in multi-objective multi-colony robot foraging. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1199–1200. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[83] Daniel J Stilwell and John S Bay. Toward the development of a material transport system using swarms of ant-like robots. In *International Conference on Robotics and Automation (ICRA)*, pages 766–771. IEEE, 1993.

[84] Niko Sünderhauf, Feras Dayoub, Sean McMahon, Ben Talbot, Ruth Schulz, Peter Corke, Gordon Wyeth, Ben Upcroft, and Michael Milford. Place categorization and semantic mapping on a mobile robot. In *International Conference on Robotics and Automation (ICRA)*, pages 5729–5736. IEEE, 2016.

[85] Niko Sünderhauf, Sareh Shirazi, Adam Jacobson, Feras Dayoub, Edward Pepperell, Ben Upcroft, and Michael Milford. Place recognition with convnet landmarks: Viewpoint-robust, condition-robust, training-free. *Proceedings of Robotics: Science and Systems XII*, 2015.

[86] J. Surber, L. Teixeira, and M. Chli. Robust visual-inertial localization with weak gps priors for repetitive uav flights. In *International Conference on Robotics and Automation (ICRA)*, pages 6300–6306, May 2017.

[87] RJ Sutherland, GL Chew, JC Baker, and RC Linggard. Some limitations on the use of distal cues in place navigation by rats. *Psychobiology*, 15(1):48–57, 1987.

[88] Jonas Svennebring and Sven Koenig. Trail-laying robots for robust terrain coverage. In *International Conference on Robotics and Automation (ICRA)*, volume 1, pages 75–82. IEEE, 2003.

[89] Tsukamoto Taisho and Tanaka Kanji. Mining dcnn landmarks for long-term visual slam. In *International Conference on Robotics and Biomimetics (ROBIO)*, pages 570–576. IEEE, 2016.

[90] Janis Tiemann, Florian Schweikowski, and Christian Wietfeld. Design of an uwb indoor-positioning system for uav navigation in gnss-denied environments. In *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–7. IEEE, 2015.

[91] Edward C Tolman. Cognitive maps in rats and men. *Psychological Review*, 55(4):189, 1948.

[92] Amirmasoud Ghasemi Toudeshki, Faraz Shamshirdar, and Richard Vaughan. Robust UAV visual teach and repeat using only sparse semantic object features. In *Canadian Conference on Computer and Robot Vision (CRV)*, pages 182–189. IEEE, 2018.

[93] Marco Trincavelli. Gas discrimination for mobile robots. *KI-Künstliche Intelligenz*, 25(4):351–354, 2011.

[94] Richard T Vaughan, Kasper Støy, Gaurav S Sukhatme, and Maja J Matarić. Blazing a trail: insect-inspired resource transportation by a robot team. In *Distributed Autonomous Robotic Systems 4*, pages 111–120. Springer, 2000.

[95] Richard T Vaughan, Kasper Støy, Gaurav S Sukhatme, and Maja J Matarić. Whistling in the dark: cooperative trail following in uncertain localization space. In *Proceedings of The Fourth International Conference on Autonomous Agents*, pages 187–194. ACM, 2000.

[96] Richard T Vaughan, Kasper Stoy, Gaurav S Sukhatme, and Maja J Mataric. Lost: Localization-space trails for robot teams. *IEEE Transactions on Robotics and Automation*, 18(5):796–812, 2002.

[97] Ranxiao Frances Wang and Elizabeth S Spelke. Human spatial representation: Insights from animals. *Trends in Cognitive Sciences*, 6(9):376–382, 2002.

[98] Tsun-Hsuan Wang, Hung-Jui Huang, Juan-Ting Lin, Chan-Wei Hu, Kuo-Hao Zeng, and Min Sun. Omnidirectional cnn for visual place recognition and navigation. In *International Conference on Robotics and Automation (ICRA)*, pages 2341–2348. IEEE, 2018.

[99] Michael Warren, Melissa Greeff, Bhavit Patel, Jack Collier, Angela P Schoellig, and Timothy D Barfoot. There's no place like home: Visual teach and repeat for emergency return of multirotor uavs during gps failure. *Robotics and Automation Letters*, 4(1):161–168, 2019.

[100] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. In *International Conference on Learning Representations (ICLR)*, 2015.

[101] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

[102] Jingwei Zhang, Lei Tai, Joschka Boedecker, Wolfram Burgard, and Ming Liu. Neural slam: Learning to explore with external memory. *arXiv preprint arXiv:1706.09520*, 2017.

[103] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *International Conference on Robotics and Automation (ICRA)*, pages 3357–3364. IEEE, 2017.