Unsupervised Learning for Robust Fitting: A Reinforcement Learning Approach

Giang Truong* Huu Le** David Suter* Erchuan Zhang* Syed Zulqarnain Gilani*
*School of Science, Edith Cowan University, Australia
**Department of Electrical Engineering, Chalmers University of Technology
{h.truong, d.suter, erchuan.zhang, s.gilani}@ecu.edu.au, huul@chalmers.se

Abstract

Robust model fitting is a core algorithm in a large number of computer vision applications. Solving this problem efficiently for datasets highly contaminated with outliers is, however, still challenging due to the underlying computational complexity. Recent literature has focused on learning-based algorithms. However, most approaches are supervised (which require a large amount of labelled training data). In this paper, we introduce a novel unsupervised learning framework that learns to directly solve robust model fitting. Unlike other methods, our work is agnostic to the underlying input features, and can be easily generalized to a wide variety of LP-type problems with quasi-convex residuals. We empirically show that our method outperforms existing unsupervised learning approaches, and achieves competitive results compared to traditional methods on several important computer vision problems[1].

1. Introduction

Many computer vision applications require the estimation of a model from a set of observations [14]. In outlier-free settings, fitting a geometric model to a dataset can be performed relatively easily by, for example, solving a least squares problem. However, in the presence of outliers in the data, a robust estimator [11, 15] must be employed to ensure the stable performance of any algorithm. As an example, consider SLAM [24], which is now a fundamental building block in several robotic or autonomous driving pipelines. It requires multiple estimations of the fundamental/essential matrices (between the consecutive views, captured along the camera trajectory). In many circumstances, erroneous correspondences between the frames could lead to incorrect camera pose estimation. Consequently, if the outliers are not removed, the whole tracking trajectory could be severely affected. Therefore, it is desirable to design robust fitting algorithms that are highly accurate and able to achieve real-time performance. This is a challenging task, as solving robust fitting optimally has been shown to be NP-hard [3, 5].

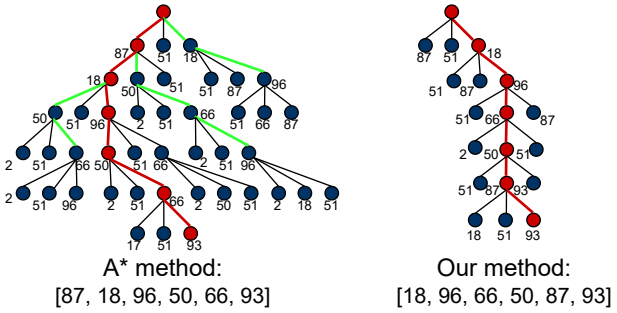


Figure 1: Illustration of the solutions found by our unsupervised learning method (right) and a globally optimal algorithm [4] (left). The number shows the specific index of points in the point set. The admissible heuristic in A* method brings the search into some fruitless subparts (green line) before discovering optimal solution (red line). Our agent learns to remove outliers by traversing from the initial state to the goal state in the minimal number of steps (the states are numbered based on the index of the removed point). Observe that both methods terminate at the same solution (i.e., both remove the same set of outliers).

In addition to popular methods such as Random Sample Consensus (RANSAC) [11] and a number of randomized or deterministic variants [7, 6, 21, 17, 2, 4, 1], the advent of deep learning in recent years has inspired research in learning-based approaches for robust estimation [30, 31, 23, 28, 8, 19]. The main idea behind these techniques is to exploit the learning capabilities of deep Convolutional Neural Networks (CNNs) to directly regress the robust estimates [19, 8], or quickly identify the outliers [23] These approaches have demonstrated their superior performance on many datasets, and hence, developing learning-based robust estimators can be a promising research direction. However, most learning techniques mentioned above

[1]Code is available at: https://github.com/hagianga21/MaxCon_RL

are supervised, hence they typically require a large amount of labelled data. This potential bottleneck could be resolved by either generating ground truth data automatically: by using synthetic data, or by using conventional methods (e.g. RANSAC) to generate ground truth. However, these "quick fixes" have their own drawbacks. Specifically, a network fully trained on synthetic data may not be able to generalize well to real world scenarios since it has not been exposed to real examples during training. Similarly, ground truth obtained from classic conventional methods is not guaranteed to be the gold standard as the obtained solutions may be incorrect. One could, on the other hand, employ some global consensus maximization methods [4] to generate the ground truth, but this would be at the cost of an exceptionally slow training process. Moreover, some methods are problem-specific, and it is non-trivial to extend them to other robust fitting tasks.

We address these problems and present a novel unsupervised learning framework for robust fitting. Inspired by the success of Reinforcement Learning (RL) in several unsupervised tasks [22, 32], we cast our robust fitting problem as a special case of goal oriented learning. Such a transformation is achieved thanks to the underlying tree structure of consensus maximization [4, 18, 1]. Moreover, we also propose a novel architecture that efficiently captures the instantaneous state of the data during transition. Fig. 1 shows an example of a 2D line fitting problem, where we plot the trajectory of A* [4] (a globally optimal algorithm), and the path traversed by our agent from the initial state to the goal state. Observe that both remove the same set of outliers, which demonstrates the learning capability of our network to effectively explore the environment. Furthermore, in contrast to the implementation of A*, which explores redundant bases before reaching the optimal, our network can quickly identify the shortest path to reach the goal state, resulting in significantly faster run times. (see Section. 3 for more detail.) To the best of our knowledge, our work is the first to learn a deep architecture model in a reinforcement learning paradigm for consensus maximization in computer vision.

Contributions The main contributions of our paper can be summarized as follows:

- We propose a novel unsupervised learning framework for robust estimation. By exploiting the *special structure* (see Section. 3) of the problem under the consensus maximization formulation, we incorporate robust model fitting into the well-known goal-oriented reinforcement learning framework, resulting in an efficient learning mechanism without any supervision.
- We develop a new state embedding scheme based on a graph convolutional network, and an efficient backbone network that allows our agent to effectively explore the action space to achieve the goal state. (see Section. 3.3 and Section. 3.4)

1.1. Related Work

Solving robust model fitting has a rich literature, where the randomized method such as RANSAC [11] is considered to be the most popular approach because it is simple to implement, and provides competitive results for many real world problems. The sampling scheme behind RANSAC has inspired many of its variants [21, 33, 6], but using them on data highly contaminated with outliers still results in unsatisfactory outcomes. In addition to the randomized variants whose purpose is to improve RANSAC, deterministic schemes have also become popular [26, 25, 20, 2]. However, their use in real-time applications is still limited due to their long processing time. On the other hand, algorithms that offer globally optimal solutions are also under active research [4, 1]. Despite their elegant algorithmic constructions, which often do improve their run time, their general long run time renders them impractical to use them for most real-world applications.

Learning to fit robust models is an interesting idea that has emerged in recent years, thanks to the learning capability of deep CNNs. Such methods have shown promising results in several fitting tasks. Ranftl et al. [30] proposed a deep network for fundamental matrix estimation by learning the weights for the residuals. Learning to solve homography estimation has also been addressed in [19]. The drawback of these methods is that they are problem-specific, hence extending to a more general class of model fitting instances is not very trivial. The use of an attention mechanism for robust feature matching has also been considered [23, 31]. The idea behind these approaches is to learn to classify correct inliers from the potentially contaminated set. Although efficient, most learning-based approaches require a sufficiently large amount of training data, which could make them impractical in new environments.

Our work is closely related to the unsupervised learning approach for consensus maximization proposed by Probst et al. [28]. However, we take a different approach by exploiting the tree structure of the underlying fitting problem. This allows us to easily train our network from scratch, while [28] requires a certain supervised signal in order to work effectively. We show that training our network from scratch in an unsupervised fashion is straightforward and at the same time the results are competitive when compared with [28].

2. Background

2.1. Problem Formulation

While there exist different ways to formulate robust estimates, our work uses the popular consensus set maximization formulation [4]. The objective is to find an estimate $\boldsymbol{\theta}^* \in \mathbb{R}^d$ that is consistent with as many of the observations $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^N$ as possible, i.e., finding the maximum number of inliers up to a predefined threshold ϵ:

$$\begin{aligned} \max_{\boldsymbol{\theta}\in\mathbb{R}^d, \mathcal{I}\subseteq\mathcal{X}} \quad & |\mathcal{I}| \\ \text{subject to} \quad & r(\mathbf{x}_i|\boldsymbol{\theta}) \leq \epsilon, \forall \mathbf{x}_i \in \mathcal{X}. \end{aligned} \tag{1}$$

where $r(\mathbf{x}_i|\boldsymbol{\theta})$ is the residual of $\mathbf{x}_i$ w.r.t model parameter $\boldsymbol{\theta}$. The solution $(\mathcal{I}^*, \boldsymbol{\theta}^*)$ of (1) provides the optimal inlier set $\mathcal{I}^*$ that is consistent with the estimate $\boldsymbol{\theta}^*$. Similar to other consensus maximization approaches, we also focus on quasi-convex residuals having the form [16],

$$r(\boldsymbol{\theta}) = \frac{p(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}, \tag{2}$$

where $p(\boldsymbol{\theta}) \geq 0$ is a convex function and $q(\boldsymbol{\theta}) > 0$ is a linear function. The objectives of several vision problems possess such quasi-convexity [16]. In this paper, to assist visualization of the mathematical concepts, most examples and analyses will be based on the linear fitting problems, whose residual functions are in fact convex, and can be written as follows: given $\mathbf{x}_i = (\mathbf{a}_i; b_i)$ with $\mathbf{a}_i \in \mathbb{R}^d$ and $b_i \in \mathbb{R}$,

$$r(\boldsymbol{\theta}) = |\mathbf{a}_i^T\boldsymbol{\theta} - b_i|. \tag{3}$$

2.2. Goal-Oriented Reinforcement Learning

Our proposed unsupervised learning approach for robust estimation is based upon the well-known goal-oriented reinforcement learning (RL) framework [32], which is briefly outlined in this subsection. This framework consists of an agent A that aims to navigate in an environment to reach a pre-defined goal, in the smallest number of steps, to maximize a total reward. Each time step is associated with a state $s^{(t)}$. The agent can select an action $a_i^{(t)} \in \mathcal{A}^{(t)}$, where $\mathcal{A}^{(t)}$ is the set of actions that are available at time t. Based on the action a taken by the agent, the environment will transition to a new state $s^{(t+1)}$ and return a reward $r_t(a)$, where the reward function $r_t(.)$ depends on the particular application. The goal of the framework is for the agent to reach the pre-defined goal that maximizes the cumulative reward (also known as return),

$$R = \sum_{t=t_0}^{\infty} \beta^{t-t_0} r_t, \tag{4}$$

obtained from the initial state to the final state, where β^{t-t_0} is the discount factor to weight the importance of the Q value at a particular time step. Commonly, deep Q learning [22] is used in most RL frameworks, for the agent to learn the optimal actions. In particular, Q learning is a model-free RL method which learns the quality of actions (for the agent to take appropriate action under a particular circumstance). Deep Q-learning is the fundamental model used in our work, and will be further outlined in Section. 3

3. Proposed Unsupervised Learning Approach

While RL has shown its strength in several applications, applying it to robust estimation is by no means a trivial task. The main challenges lie in the definition of: a state, reward function, goal specification and the design of an agent that can learn to efficiently explore the environment in an optimal way. In this section, we introduce a novel framework that enables the use of RL for our robust fitting problem.

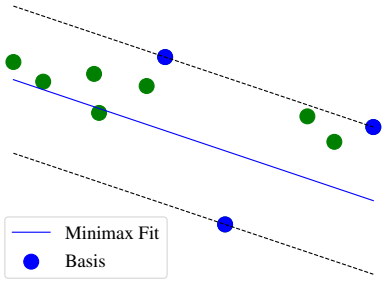


Figure 2: Illustration of a minimax fitting problem for a set of points in 2D. Blue dots represent measurements with largest residual, which form the *basis set*.

3.1. Definitions

Given a set of measurements (observations) $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^N$, let us first consider the minimax fitting problem that returns the estimate which minimizes the maximum residual:

$$f(\mathcal{X}) = \min_{\boldsymbol{\theta}\in\mathbb{R}^d, \gamma\in R} \gamma, \quad \text{s.t. } r(\mathbf{x}_i|\boldsymbol{\theta}) \leq \gamma \;\; \forall i, \mathbf{x}_i \in \mathcal{X}. \tag{5}$$

It can be proven that if the residual function $r(\mathbf{x}_i|\boldsymbol{\theta})$ is quasi-convex, the above problem is also quasi-convex, hence it can be solved efficiently up to global optimality using any off-the-shelf solver [16]. Fig. 2 shows an example of the minimax fit of a set of points in 2D. This problem is the core sub-problem in our learning framework.

Observe that if the optimal solution γ^* obtained from solving the above problem is not greater than the inlier threshold, i.e., $\gamma^* \leq \epsilon$, then the solution $\boldsymbol{\theta}^*$ obtained from (5) also solves the robust fitting problem (1) (there are no outliers). Otherwise, the optimal consensus must be a subset $\mathcal{I}^*$ such that $f(\mathcal{I}^*) \leq \epsilon$. Therefore, the goal of our agent is to gradually remove a subset of outliers to reach the subset $\mathcal{I}^*$. Obviously, in order to maximize $\mathcal{I}^*$, the number of outliers removed by our agent must be minimized.

State. Under the RL framework, let us consider associating each subset $\mathcal{S} \subseteq \mathcal{X}$ with a state $s_\mathcal{S}$ (the detailed construction of states for our network will be discussed in the following sections), and set $s_\mathcal{X}$ to the initial state at which our agent will start the exploration process, i.e., $s^{(t=0)} = s_\mathcal{X}$. From the above discussion, at a particular state $s_\mathcal{S}$, if the action taken by our agent is to remove one data point $\mathbf{x}_j \in \mathcal{S}$ such that $f(\mathcal{S} \setminus \mathbf{x}_j) < f(\mathcal{S})$, then the task of the agent is to find the state $s_\mathcal{I}$ such that $f(\mathcal{I}) \leq \epsilon$ in the smallest number of steps (i.e., to minimize the number of outliers removed). Refer to Fig. 3 for a visualization of a state and its associated actions for a 2D fitting problem.

Goal and Reward Function. We therefore define our goal state, based on the RL framework discussed in the previous section, as a state $s_\mathcal{S}$ such that $f(\mathcal{S}) \leq \epsilon$. Also, we can now define a reward function $e(.)$ associated with a state $s_\mathcal{S}$ to be

$$e(s_\mathcal{S}) = \begin{cases} 0 & \text{if } f(\mathcal{S}) \leq \epsilon \\ -1 & \text{otherwise.} \end{cases} \tag{6}$$

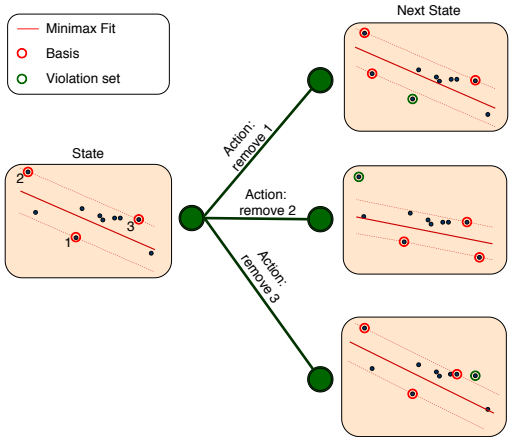


Figure 3: Visualization of states and actions. The initial state corresponds to the minimax fit of the original point set. The action, associated a particular state, consists of removing a point in the basis set and conduct minimax fit for the remaining points.

As can be observed, the maximized total reward obtained by our agent, using reward function (6), corresponds to moving from the initial state to the goal state in the minimum number of steps.

3.2. Generating Action Sets

As previously discussed, the available actions associated with a particular state $s_{\mathcal{S}}$ corresponds to the removal of points $\mathbf{x}_j \in \mathcal{S}$ such that $f(\mathcal{S} \setminus \mathbf{x}_j) < f(\mathcal{S})$. One could test all the points in $\mathcal{S}$ to generate the action set. However, such exhaustive testing turns out to be unnecessary for our problem. To reduce the number of available actions at each state, we exploit a special property of our application as follows.

Assume $\boldsymbol{\theta}_{\mathcal{S}}$ is the solution of (5) for a set $\mathcal{S}$. Let us consider the set $\mathcal{B}_{\mathcal{S}}$ containing points having the largest residuals,

$$\mathcal{B}_{\mathcal{S}} = \{\mathbf{x}_j \in \mathcal{S} | r(\mathbf{x}_j | \boldsymbol{\theta}_{\mathcal{S}}) = f(\mathcal{S})\} \tag{7}$$

Following the terminology in [4, 1], we also call $\mathcal{B}_{\mathcal{S}}$ a basis of the set $\mathcal{S}$. In the example shown in Figure 2, the basis for a 2D-line fitting problems w.r.t. to the current estimate are the points plotted in blue.

Given $\mathcal{B}_{\mathcal{S}}$, one can prove (see [4, 1]) that,

$$f(\mathcal{S} \setminus \mathbf{x}_j) < f(\mathcal{S}), \quad \forall \mathbf{x}_j \in \mathcal{B}_{\mathcal{S}}. \tag{8}$$

Intuitively, removing a point belonging to the basis set guarantees the reduction of the minimax fit for the remaining set (see Figure 3). This suggests that the actions associated with a state $s_{\mathcal{S}}$ corresponds to the removal of a point in $\mathcal{B}_{\mathcal{S}}$, and conduct minimax fit for the remaining points. Figure 3 visually illustrates how the actions can be generated from a particular state.

Moreover, for quasi-convex residuals in d dimension, the maximum size of $\mathcal{B}_{\mathcal{S}}$ is proven to be $|\mathcal{B}_{\mathcal{S}}| \leq d+1$(see [9]). Therefore, at a particular state, the maximum number of available actions in our cases is $d+1$. This property significantly reduces the action space for our learning framework, and is one of the key factors that leads to the efficacy of our learning scheme.

3.3. State Encoding

In order to use the states above as the inputs to our network, they first need to be properly encoded. Based on the above discussion, the crucial information for a state $s_{\mathcal{S}}$ consist of the point set $\mathcal{S}$ together with its basis $\mathcal{B}_{\mathcal{S}}$ (which also stores the information about the available actions associated with $s_{\mathcal{S}}$). To enrich the information for $s_{\mathcal{S}}$, our state encoding also comprises the set $\mathcal{V}_{\mathcal{S}} = \mathcal{X} \setminus \mathcal{S}$. Clearly, $\mathcal{V}_{\mathcal{S}}$ encodes the agent's state traversal history before reaching the state $s_{\mathcal{S}}$. Given $\mathcal{S}$, $\mathcal{B}_{\mathcal{S}}$ and $\mathcal{V}_{\mathcal{S}}$, we construct a matrix $\mathbf{S}_{\mathcal{S}}$ to feed it into our network.

To encode $\mathcal{B}_{\mathcal{S}}$ and $\mathcal{V}_{\mathcal{S}}$, we define two binary vectors $\mathbf{b}_{\mathcal{S}} \in \{-1, 1\}^N$ and $\mathbf{v}_{\mathcal{S}} \in \{-1, 1\}^N$, respectively, defined as follows (we use the notation $\mathbf{x}[i]$ to denote the ith component of a vector $\mathbf{x}$),

$$\mathbf{b}_{\mathcal{S}}[i] = \begin{cases} 1 & \text{if } \mathbf{x}_i \in \mathcal{B}_{\mathcal{S}} \\ -1 & \text{otherwise.} \end{cases} \quad \text{and } \mathbf{v}_{\mathcal{S}}[i] = \begin{cases} 1 & \text{if } \mathbf{x}_i \in \mathcal{V}_{\mathcal{S}} \\ -1 & \text{otherwise.} \end{cases} \tag{9}$$

Therefore, each state $s_{\mathcal{S}}$ can now be encoded by the matrix $\mathbf{S}_{\mathcal{S}}$,

$$\mathbf{S}_{\mathcal{S}} = [\mathbf{H} \;\; \mathbf{b}_{\mathcal{S}} \;\; \mathbf{v}_{\mathcal{S}}], \tag{10}$$

where H is the matrix that collects all the data points in the input set $\mathcal{X}$. More specifically, we set the i-th row of the matrix $\mathbf{H}$ to $h(\mathbf{x}_i)$, where $h(.)$ being any mapping that can well represent the information of a given input $\mathbf{x}_i$. For example, in linear fitting, $h\,(\mathbf{x}_i = (\mathbf{a}_i, b_i))$ can be simply chosen to be $h(\mathbf{x}_i) = [\mathbf{a}_i^T \; b_i]$ (i.e., h concatenates $\mathbf{a}_i$ and b_i to make a row vector).

Note that since our state involves point sets, one expects our network to be permutation invariant w.r.t. the input $\mathcal{S}$. In other words, changing positions of the rows in the matrix $\mathbf{S}_{\mathcal{S}}$ does not affect the output of our network. Such permutation invariance can be achieved by using the graph CNN architecture, described in the following section.

3.4. Network (Agent) Design

Similar to other RL problems, our agent needs to learn to traverse from the initial state, to the goal state, in such a way that the total rewards is maximized (in the smallest number of steps). This section describes our agent design, which is also illustrated in Figure 4.

The network takes as input a particular state (encoding using the method described in Section 3.3), and outputs the predicted rewards for the actions associated with the input state. We use Θ to represent the network parameters, we denote by $\hat{Q}(s^{(t)}, a|\Theta)$ the action-value function that returns the optimal reward if the action a is taken given the current state $s^{(t)}$. In other words, the selected action given a current

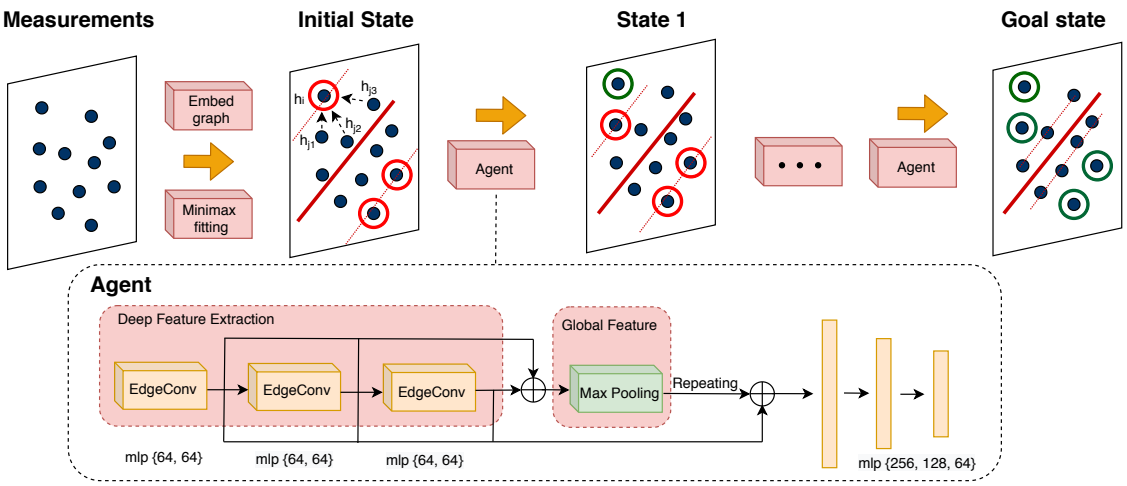


Figure 4: Illustration of our proposed framework. Top row shows the (unrolled) operations (we use an instance of 2D line fitting as an example). Given a set of measurements, minimax is performed to obtain the initial state. Then, the state is encoded into a graph representation which is fed to the agent to predict the expected Q value (returns) of choosing each point in basis to eliminate. The agent then performs an action, receives a reward and moves to the next state. This process iterates until the agent reaches the goal state. Bottom row depicts the design of our network (agent).

state $s^{(t)}$ is the action that maximizes $\hat{Q}$, i.e.,

$$a = \underset{a \in \mathcal{A}(\mathcal{B}_S)}{\arg\max}\, \hat{Q}(s^{(t)}, a|\Theta), \tag{11}$$

where we use $\mathcal{A}(\mathcal{B}_S)$ to denote the set of actions associated with the removals of points in the basis set $\mathcal{B}_S$ as described in Section 3.2.

In order to achieve permutation invariance of the input, we design the first stage of our network (shown in the Deep Feature Extraction (DFE) block in Figure 4) to be a series of Edge Convolution (EdgeConv) Layers [34], which was originally inspired from PointNet [29]. We employed the EdgeConv layer because, unlike PointNet, it has the ability to capture local geometric structures, by taking into account the nearest neighbors of each single input (interested readers are referred to [34] for more detail), hence more information can be extracted to improve the learning capability. The main role of this DFE block is to capture the relationship between every single input data point and its local geometric structure. Then, the 'global set feature' is obtained from the Global Feature block shown in Figure 4. The repeated concatenation of the global feature with the individual input features is then fed in to a multi-layer perceptron (MLP) to obtain the expected rewards. We then apply a mask to extract only rewards for points in $\mathcal{B}_S$.

3.5. Learning algorithm

Based on the components discussed above, this section introduces our general learning algorithm, which is summarized in Algorithm 1. Our learning framework relies on the popular deep Q-learning approach that has been used extensively in several other RL applications. The training is repeated over multiple episodes. At the start of each episode, a set of measurements $\mathcal{X}$ containing N data points (with N fixed throughout the training process) is randomly sampled from the training set. Note that the outlier rates in $\mathcal{X}$ is randomly chosen to be in the range from 1% to 40% for each episode. A minimax fit (5) is then performed on $\mathcal{X}$ to obtain the initial state $s_{\mathcal{X}}$. Starting from $s_{\mathcal{X}}$, our agent explores the search space by passing through multiple states until reaching the goal state. During the training process, the action taken at each state is sampled based on the popular ε - greedy policy [22]. After taking an action, the agent receives the reward computed based on (6) and moves to next state. The the network parameters are then updated based on the well-known Bellman's equation [22],

$$\hat{Q}(s^{(t)}, a|\Theta) = e(s^t) + \gamma \max_{a \in \mathcal{A}(\mathcal{B}_S)} \hat{Q}(s^{(t+1)}, a|\Theta). \tag{12}$$

Therefore, the network parameters are updated by minimizing the temporal difference error δ defined by

$$\delta = \rho(e(s^t) + \gamma \max_{a \in \mathcal{A}(\mathcal{B}_S)} \hat{Q}(s^{(t+1)}, a|\Theta) - \hat{Q}(s^{(t)}, a|\Theta)), \tag{13}$$

where we choose ρ to be the Huber loss [15]. When the agent reaches the goal state (i.e., $f(\mathcal{S}) \leq \epsilon$), another set of measurements $\mathcal{X}$ is taken and a new episode is started. As we use the popular PyTorch framework to implement our network, the optimization of (13) to update the network parameters can be performed by the off-the-shelf gradient-

Algorithm 1 Main algorithm.

Initialize experience relay memory $\mathcal{M}$
for episode e = 1 to L **do**
  Take a set of putative measurements $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^{N}$
  Obtain maximum residual $f(\mathcal{S})$ and basis $\mathcal{B}_{\mathcal{S}}$ by solving (5)
  Initialize first state $s^{(t=0)}$
  while $(f(\mathcal{S}) > \epsilon)$ **do**

$$a_t = \begin{cases} \text{random action } a \in \mathcal{A}(\mathcal{B}_{\mathcal{S}}), & \text{w.p.} \varepsilon \\ \arg\max_{a \in \mathcal{A}(\mathcal{B}_{\mathcal{S}})} \hat{Q}(s^{(t)}, a|\Theta), & \text{otherwise} \end{cases}$$

    Get reward $e(s^t)$) and move to next state s^{t+1}
    Add tuple $(s^t, a^t, s^{t+1}, e(s^t))$ to $\mathcal{M}$
    Sample random batch from $\mathcal{M}$
    Update network parameter Θ
  end while
end for

Algorithm 2 Local Tree Refinement

Require: Input data $\mathcal{X}$, initial solution $\mathcal{S}^0$
$t \leftarrow 0$, `improved` $\leftarrow$ True
while `improved` **do**
  $\hat{\mathcal{S}}^{(t)} \leftarrow \mathcal{X} \setminus \mathcal{S}^{(t)}$, `improved` $\leftarrow$ False
  for $\mathbf{x}_j \in \hat{\mathcal{S}}^{(t)}$ **do**
    if $f(\mathcal{S}^{(t)} \cup \mathbf{x}_j) \leq \epsilon$ **then**
      $\mathcal{S}^{(t)} \leftarrow \mathcal{S}^{(t)} \cup \mathbf{x}_j$; `improved` $\leftarrow$ True.
    end if
  end for
  $t \leftarrow t + 1$.
end while

based solvers. More information about the choices of training parameters can be found in the supplementary material.

3.6. Local Tree Refinement

Recall from previous sections that the solution returned by our network is a set $\mathcal{S}^*$ such that $f(\mathcal{S}^*) \leq \epsilon$. Since the proposed algorithm is sub-optimal, the consensus size obtained from $\mathcal{S}^*$ could be less than the optimal solution $\mathcal{I}^*$, i.e., $|\mathcal{S}^*| \leq |\mathcal{I}^*|$, where $\mathcal{I}^*$ is the solution of Problem (1). To partially overcome this, we propose a simple heuristic in order to gradually improve our obtained solution, which is summarized in Algorithm 2. Intuitively, starting from the initial solution $\mathcal{S}^{t_1=0} = \mathcal{S}^*$ (note that we use t_1 to avoid confusion with the state index t used in the previous sections), we test all points $\mathbf{x}_j$ in the current outlier set $\hat{\mathcal{S}}^{t_1} = \mathcal{X} \setminus \mathcal{S}^{t_1}$ and add points that lead to consensus size improvement, i.e., $f(\mathcal{S}^{t_1} \cup \mathbf{x}_J) \leq \epsilon$. This process is repeated until no more points can be added.

4. Experiments

All experiments were executed on an Ubuntu machine with an Intel Core 3.70GHz i7 CPU, 32Gb RAM and a Geforce GTX 1080Ti GPU. Our network is implemented in Python using the popular PyTorch [27] and Torch Geometric [10]. We present the main results in this section. A keen reader can refer to the supplementary material for implementation details.

Baseline Algorithms. We compare our method against the following: Original A* tree search [4], A*- Non-Adjacent Path Avoidance (NAPA) with Dimension Insensitive Branch Pruning (DIBP) [1], RANSAC [12], and Local Optimization for RANSAC (LO-RANSAC) [21]. In addition, we also run a random baseline (RB) approach, where at each state, the agent takes an action randomly until reaching a goal state. The objective of comparing with RB is to show that our network traverses the path intelligently as opposed to taking random guesses. For fundamental matrix estimation experiments, we also compare our method with the state-of-the-art unsupervised approach for consensus maximization introduced by Probst et al. [28][2] (since only the source code for fundamental estimation is publicly available, we did not compare against [28] in the line and plane fitting experiments).

In order to have a fair comparison, for each problem instance, we allowed the RANSAC variants to take equal (or longer) run times compared to the run time required by our method.

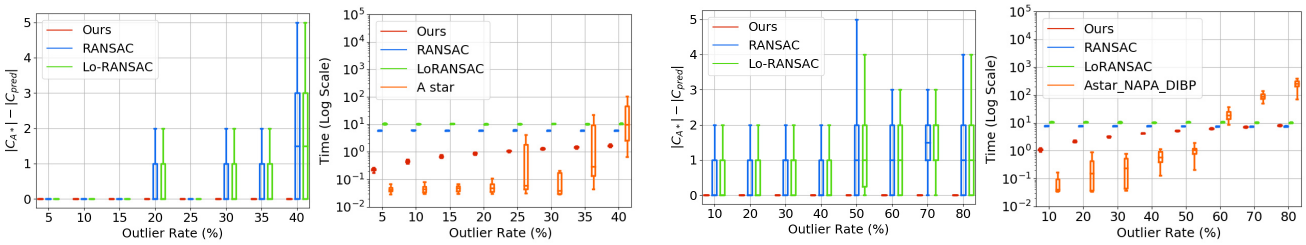
4.1. Robust Line And Plane Fitting

We first test the algorithms on robust 2D and 3D linear fitting with synthetic data (with various outlier rates) in order to evaluate their performance in a well-controlled setting.

Robust 2D Line Fitting. To create data (for both training and testing), we randomly generate N (where N is chosen to be 100 and 200) points $a_i \in \mathbb{R}$, and a line parameter $\hat{\boldsymbol{\theta}} \in \mathbb{R}^2$. From $\{a_i\}$ and $\hat{\boldsymbol{\theta}}$, we obtain the set $b_i \in \mathbb{R}$ by computing $b_i = a_i\theta_1 + \theta_2$. Of the N points in total, N_o are randomly chosen to be outliers by corrupting their b_i with a uniformly distributed noise between $[-5, -0.1) \cup (0.1, 5]$, while the remaining $N - N_o$ points are perturbed with a uniform noise in the range $[-0.1, 0.1]$. The inlier threshold is set to $\epsilon = 0.1$. When training, we vary the outlier rate from 1 to 40%. We compare the performance of our network against the random baseline models, and Original A*. Our method and all randomized methods were executed 100 times.

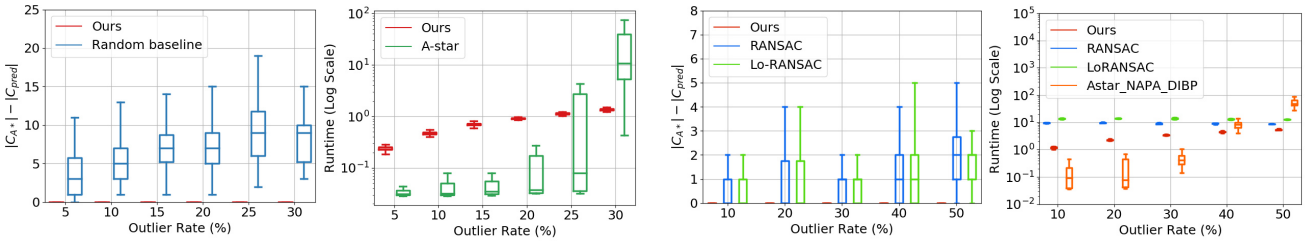
We report our results in the form of boxplots to summarize important statistical information, including the median, the variance, and the extreme cases of the data. Since optimal solutions from A* are available in this experiment,

[2] We used the source code provided by the authors with default settings



(a) $N = 100$ (b) $N = 200$

Figure 5: 2D line fitting on (a) 100 and (b) 200 points, with various outlier rates. Left: Distribution of distance between predicted consensus and global solution (obtained using A*) with baseline models. Right: Run-time(s) of our method compared to optimal methods (A*).



(a) $N = 100$ (b) $N = 200$

Figure 6: 3D plane fitting on (a) 100 and (b) 200 points, with various outlier rate. Left: Distribution of distance between predicted consensus and global solution (obtained using A*) with random baseline model. Right: Run-time(s) of our method compared to the optimal method (A*).

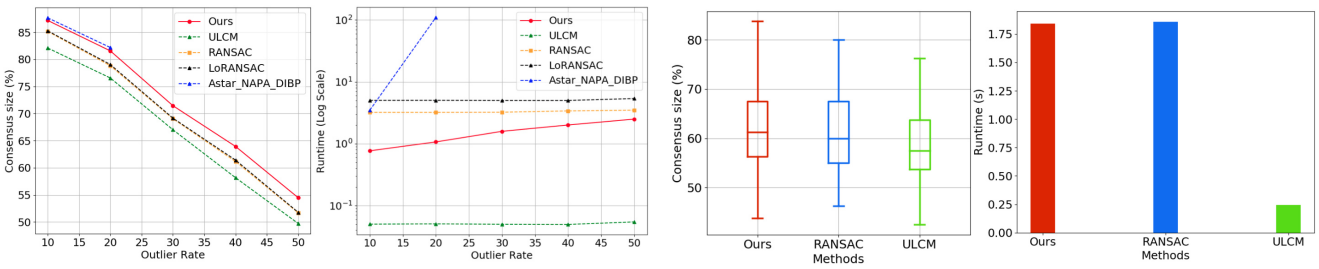
we use them as the gold standard to test the quality of the obtained consensus sets. The box plot in Fig. 5a (left) depicts the differences in the consensus sizes between A* and other methods for $N = 100$ points. As shown, our method consistently obtains almost optimal solutions, while the solutions provided by RANSAC and LO-RANSAC are rather unstable over the runs (and unsatisfactory for higher outlier rates). Fig. 5a (right) shows the run times for all methods. Observe that, compared to A*, our run times are an order-of-magnitude faster. Also note that the run time of A*-NAPA-BIDP [1] increases exponentially with higher outlier rate. On the other hand, although we allow RANSAC and LO-RANSAC [21] to run longer than ours, we still achieve much better results. We verify the robustness and generalization of our framework by testing it with double the number of points ($N = 200$) and higher outlier rates (up to 80%) without retraining the model. The same conclusions can be drawn from Fig. 5b.

Robust 3D Plane Fitting. A setting similar to the 2D Fitting experiment (above) is repeated for 3D plane fitting, which also has many practical applications in computer vision. Fig. 6 shows the performance comparison between our proposed method and a simple random baseline model for $N = 100$ and $N = 200$. One can see that our proposed method actually learns something, compared to a random method that has "no knowledge". We also verify the robustness and generalization of our framework by testing it with double the number of points ($N = 200$) and higher outlier rate without retraining the model. On average, our proposed method usually achieves optimal solution with a much faster run-time, compared to the A* methods. Moreover, as shown in Fig. 6, given a less time budget, our model returns optimal solutions that RANSAC-based approaches never return.

4.2. Linearized Fundamental Matrix Estimation

We also test the algorithms on fundamental matrix estimation with linearized residuals, following the settings used in state-of-the-art [4, 2]. For this experiment, we also compare our algorithm with the state-of-the-art unsupervised learning approach ULCM [28]. It is important to highlight the fact that in these experiments it is very difficult to obtain the ground truth, as it is impractical to apply the global optimal methods (like A* and its derivatives), since they take took too long to arrive at an optimal solution. Therefore, we use the consensus size as the evaluation metric.

ModelNet40 Dataset First, we use data from the ModelNet40 [35] dataset to train and evaluate our model. Two-view pairs are generated by projecting the point clouds on two image planes having different focal lengths and extrinsic camera parameters. For each image pair, $N = 100$ correspondences with outlier rates ranging from 10% to 50% are generated for training and testing. The inlier threshold is chosen as $\epsilon = 0.1$. The left plot in Fig. 7a compares the



(a) ModelNet40 Dataset. (b) KITTI Dataset.

Figure 7: Linearized Fundamental Matrix Estimation. Left: Consensus size comparison. Right: Run time(s) in log scale.

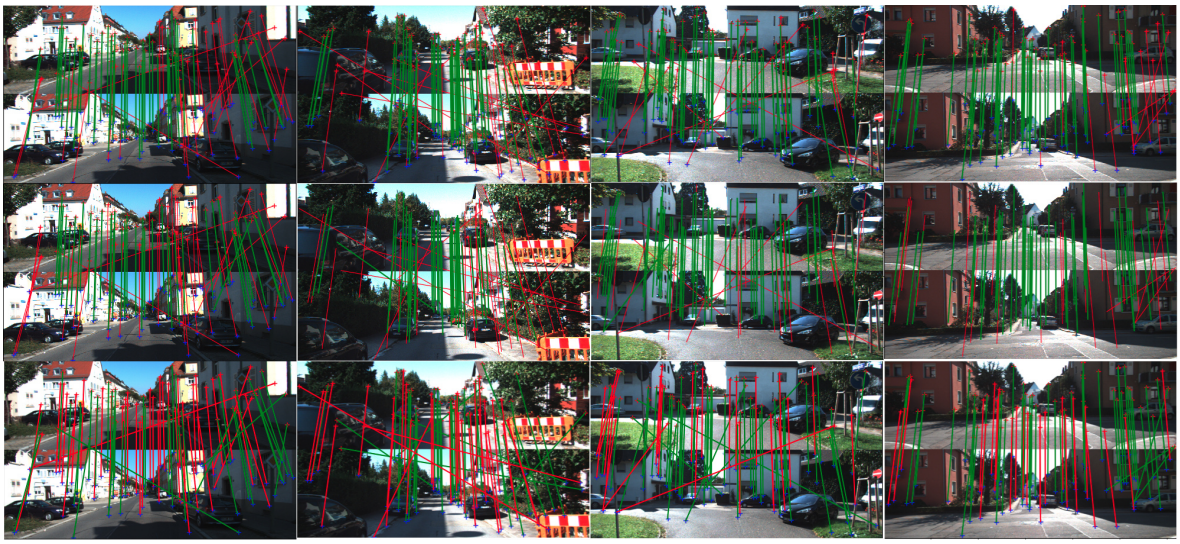


Figure 8: Qualitative results for robust fundamental matrix. Top: Ours; Middle: RANSAC; Bottom: UCLM

average consensus size on various outlier rate. The run time of A*-NAPA-DIBP [1] increases exponentially with higher outlier rates. On average, with less time budget, we generally obtain higher consensus sets, compared to RANSAC-based methods. Although the state-of-the-art unsupervised deep learning method, ULCM [28] might be comparatively inexpensive, our method is still able to perform better and achieve higher consensus size.

KITTI Dataset We repeat the robust fundamental fitting experiment on one sequence of KITTI dataset [13]. We compute and match SIFT keypoints using the VLFeat toolbox[3]. The inlier threshold is chosen as $\epsilon = 0.01$. Fig. 7b plots the consensus sizes obtained by our method in comparison with RANSAC and ULCM [28].

As seen, our method generally achieves higher consensus sizes compared to ULCM [28]. When compared to RANSAC, although our spread is larger, we have a smaller variance and higher median consensus size. This means that our method only occasionally returns the worst extreme case. We believe that our method can be improved in future work by designing better network architectures to address such outlying scenarios. We also visualize the solutions qualitatively in Fig. 8 for the three methods. The visualization is consistent with the results shown in Fig. 7b, where our method achieves inlier sets with higher quality.

5. Conclusion

We have presented in this work a novel unsupervised reinforcement learning framework for robust estimation. By exploiting the problem structure, we propose an efficient state encoding, and a back-bone network, that enables our agent to effectively learn to explore the search tree. Our approach can be applied in many traditional pipelines (e.g. Structure from Motion), which require the estimation of pairwise fundamental or homography matrices. In future, our work can be extended to other robust fitting tasks in SfM such as registration, 2D-3D matching, etc. and with larger scale datasets.

Acknowledgments David Suter and Erchuan Zhang acknowledge partial funding for this work under Australian Research Council grant DP200103448. Huu Le was partially supported by Wallenberg AI, Autonomous Systems and Software Program (WASP) and the Chalmers AI Research Center (CHAIR) Seed Projects 2020.

[3] http://vlfeat.org

References

[1] Zhipeng Cai, Tat-Jun Chin, and Vladlen Koltun. Consensus maximization tree search revisited. In *Computer Vision, 2019. ICCV 2019*, 2019. 1, 2, 4, 6, 7, 8

[2] Zhipeng Cai, Tat-Jun Chin, Huu Le, and David Suter. Deterministic consensus maximization with biconvex programming. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 685–700, 2018. 1, 2, 7

[3] Tat-Jun Chin, Zhipeng Cai, and Frank Neumann. Robust fitting in computer vision: Easy or hard? In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 701–716, 2018. 1

[4] Tat-Jun Chin, Pulak Purkait, Anders Eriksson, and David Suter. Efficient globally optimal consensus maximisation with tree search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2413–2421, 2015. 1, 2, 4, 6, 7

[5] Tat-Jun Chin and David Suter. *The Maximum Consensus Problem: Recent Algorithmic Advances*. Morgan & Claypool Publishers, 2017. 1

[6] Ondrej Chum and Jiri Matas. Matching with prosac-progressive sample consensus. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 220–226. IEEE, 2005. 1, 2

[7] Ondřej Chum, Jiří Matas, and Josef Kittler. Locally optimized ransac. In *DAGM*. Springer, 2003. 1

[8] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Deep image homography estimation. *arXiv preprint arXiv:1606.03798*, 2016. 1

[9] David Eppstein. Quasiconvex programming. *Combinatorial and Computational Geometry*, 52(287-331):3, 2005. 4

[10] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 6

[11] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 1, 2

[12] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981. 6

[13] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012. 8

[14] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003. 1

[15] Peter J Huber et al. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964. 1, 5

[16] Qifa Ke and Takeo Kanade. Quasiconvex optimization for robust geometric reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(10):1834–1847, 2007. 3

[17] Huu Le, Tat-Jun Chin, and David Suter. An exact penalty method for locally convergent maximum consensus. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE, 2017. 1

[18] Huu Le, Tat-Jun Chin, and David Suter. Ratsac-random tree sampling for maximum consensus estimation. In *2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–8. IEEE, 2017. 2

[19] Hoang Le, Feng Liu, Shu Zhang, and Aseem Agarwala. Deep homography estimation for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7652–7661, 2020. 1, 2

[20] D Khuê Lê-Huu and Nikos Paragios. Alternating direction graph matching. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4914–4922. IEEE, 2017. 2

[21] K Lebeda, J Matas, and O Chum. Fixing the locally optimized ransac. In *British machine vision conference, 2012*, 2012. 1, 2, 6, 7

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 2, 3, 5

[23] Kwang Moo Yi, Eduard Trulls, Yuki Ono, Vincent Lepetit, Mathieu Salzmann, and Pascal Fua. Learning to find good correspondences. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2666–2674, 2018. 1, 2

[24] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015. 1

[25] Carl Olsson, Olof Enqvist, and Fredrik Kahl. A polynomial-time bound for matching and registration with outliers. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 2

[26] Carl Olsson, Anders P Eriksson, and Richard Hartley. Outlier removal using duality. In *IEEE Int. Conf. on Copmuter Vision and Pattern Recognition*, pages 1450–1457. IEEE Computer Society, 2010. 2

[27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019. 6

[28] Thomas Probst, Danda Pani Paudel, Ajad Chhatkuli, and Luc Van Gool. Unsupervised learning of consensus maximization for 3d vision problems. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 929–938, 2019. 1, 2, 6, 7, 8

[29] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017. 5

[30] René Ranftl and Vladlen Koltun. Deep fundamental matrix estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 284–299, 2018. 1, 2

[31] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4938–4947, 2020. 1, 2

[32] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 2, 3

[33] Philip HS Torr and Andrew Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer vision and image understanding*, 78(1):138–156, 2000. 2

[34] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019. 5

[35] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015. 7