Data driven strategies for Active Monocular SLAM using Inverse Reinforcement Learning

Initial writeup

ABSTRACT

Learning a complex task such as low-level robot manoeuvres while preventing failure of Monocular SLAM is a challenging problem for both robots and humans. The data-driven identification of basic motion strategies in preventing Monocular SLAM failure is a largely unexplored problem. In this paper, we devise a computational model for representing and inferring strategies, formulated as Markov decision processes, where the reward function models the goal of the task as well as information about the strategy. We show how this reward function can be learnt from expert demonstrations using inverse reinforcement learning. The resulting framework allows one to identify the way in which a few chosen parameters affect the quality of Monocular SLAM estimates. The estimated reward function was able to capture expert demonstration information and the inherent expert strategy and it was possible to give an intuitive explanation to the obtained reward structure. A significant improvement in performance as compared to an intuitive hand-crafted reward function is also shown.

Keywords

Active Monocular SLAM, Inverse Reinforcement Learning, Robot navigation, Learning from Demonstration.

1. INTRODUCTION

Simultaneous Planning, Localization and Mapping (SPLAM) or Active Simultaneous Localization and Mapping (Active SLAM) [6, 7, 8, 9, 10], deals with the generation of control commands for a robot to move in an unknown environment such that the uncertainty of the localization and/or the mapping is minimized. In the past, this problem has been formulated using Model Predictive Control [7] or in an Information Gain paradigm [8, 10]. Belief-space planning has been shown to perform well in generating control commands in a continuous domain [9].

All of the works discussed above assume the availability of immediate dense range data and associated map uncertainty estimates, both of which are not directly available in a Monocular SLAM setting. But, the usage and research on Monocular SLAM is important because the hardware needed to implement it is much simpler, cheaper and physically smaller than other systems. Monocular SLAM systems have unique abilities that are not experienced with the other range based SLAM modules, they tend to break down when in the presence rotated in place, when the viewpoint changes are large, or with improper initialization of the camera pose.

There is a limited amount of work in literature that tackles SPLAM in a Monocular setting. A Next-Best-View (NBV) approach [12] to the problem showed good results for goal based trajectory planning as well as strategic exploration on Micro Aerial Vehicles (MAVs), but one may not always have the same degrees of freedom in the context of ground vehicles. Recent work shows the use of Reinforcement Learning [13] to achieve the same task on a ground robot.

Both the above mentioned methods are hand crafted methods using human intuition to model the parameters used. This may not be able to accurately capture the importance of these parameters.

To solve this problem we first model the expert policy inspired by human intuition in a principled manner through Reinforcement Learning (RL). Subsequently with the RL generated policy as the expert policy we formulate an Inverse Reinforcement Learning (IRL) model, that successively learns the rewards whose behaviour evaluates even more favourably than the expert's RL policy. This performance improvement over the expert's policy is presented in the form of minimum steps to breakage and the number of successful runs on different maps. Specifically the paper contributes in the following ways:

1. It proposes a novel IRL formulation for Active localization/SPLAM that learns SLAM safe trajectories which result in an increase in SLAM longevity before breakdown and enables effective automated Monocular SLAM.

2. The proposed IRL formulation significantly improves SLAM estimates and compares more favourably vis-a-vis an RL generated expert policy with hand crafted reward structure. in maps of varying difficulties the IRL based planner reaches distinct goal locations autonomously without SLAM breakdown a distinctly large number of times than an RL based agent.

3. The paper also reveals in a principled manner the features that affect the quality of SLAM estimates the most and the least.

The IRL problem was originally formulated within the MDP framework by [1]. Many researches provided further refinements in order to improve the original algorithms sug-

<Left blank later addition

gested by [1] and [2]. For example, [3] suggested a max-margin planning approach.[4] modeled the uncertainties involved as probabilities where the demonstrations are treated as evidence of the unknown reward function. A recent review of IRL algorithms can be found in [5].

However, most IRL approaches rely on a given model of the environment or assume that it can be accurately learned from the demonstrations. The reward function is found by first computing a policy that optimizes a reward function for an initial weight vector ω. Subsequently, the expected feature expectation of the new policy $\mu(\pi)$ can be computed. Based on this feature expectation, a new weight vector that separates the values of the expert feature expectation $\mu(\pi^E)$ and the feature expectation of the current policy $\mu(\pi)$ can be computed. These steps are repeated until the weight vector converges. Generally, a model of the dynamics is used to iteratively generate optimal trajectories under different reward functions until the generated trajectories match the ones provided by the expert.

The main focus of the paper is not to introduce new IRL methods for solving problems such as the one tackled in this paper, but rather to apply existing methods to solve a challenging problem. Through this paper we will answer the following questions:

1. Can we capture the expert demonstration data into a vector known as the reward function using Inverse reinforcement learning?
2. Is it possible to learn the expert demonstrated behaviour by solving an MDP with the obtained reward function?
3. Which of the features used affect the Monocular SLAM estimates the most and the least?

In the remainder of this paper, we will proceed as follows. In Sect. 2 and Sect. 3, we present the theoretical background for modeling decision processes, including MDPs and the IRL algorithms used. We present the experimental setup and evaluations in Sect. 4. In Sect. 5, we summarize our approach and the results.

2. BACKGROUND

As discussed in the introduction, we use Inverse reinforcement learning in a model free context to construct a reward function which advocates the expert demonstrated behaviour and then learn the behaviour of an agent by solving an MDP using the obtained reward function. Here we will first discuss about the Monocular SLAM framework and subsequently introduce the notation and basic elements necessary for the Monocular camera on a mobile robot setting including the MDP formulation.

2.1 Monocular SLAM

Simultaneous Localization and Mapping (SLAM) refers to the problem of estimating the pose and trajectory of a robot while mapping its environment at the same time. Monocular SLAM is SLAM performed using a single Monocular camera as the sole sensory input. A real time Monocular SLAM algorithm requires:

- Robust initialization of the camera pose and map.
- Accurate mapping of real world points among a subset of frames seen (keyframes).
- Reliable tracking of points over multiple frames to estimate camera trajectory.

PTAM (Parallel Tracking and Mapping) [15] was one of the first such system developed by Klein and Murray. As the name suggests, it parallely maps feature points from the world and tracks them over multiple frames to provide a 6 DoF pose estimate. **ORB-SLAM** [16] is a similar feature based Monocular SLAM that builds on the main ideas of PTAM. It uses different features for tracking as compared to PTAM and incorporates real time loop closure along with mapping and tracking. **LSD-SLAM** [17] is a state-of-the-art Monocular SLAM system that, unlike the methods mentioned above, creates a much denser 3D reconstruction of the environment. Rather than working with features, it works directly with image intensity differences. It tracks using the image as a whole, rather than some abstraction in the form of features. Mapping is done using pixel depth estimation rather than feature depth estimation.

While these methods have shown effective SLAM capabilities, feature based Monocular SLAM methods are susceptible to errors in pose estimates due to insufficient feature tracking or motion induced errors such as in-place rotation or steep change in heading angle. Motion-induced errors can be mitigated to an extent by severely restricting robot motion, but this may lead to inefficiencies in navigation. Moreover, literature that has used these to autonomously navigate with stable pose estimates are sparse.

2.2 Markov Decision Process

In order to employ IRL, the problem at hand needs to be modelled as a Markov Decision Process(MDP). In formal notations an MDP is a tuple (S, A, T, γ, R), where

- S is a set of states.
- A is a set of actions.
- $T = \{P_{sa}\}$ is a set of state transition probabilities. P_{sa} is the state transition probability distribution by taking action a from state s.
- γ is the discount factor. $\gamma \in [0, 1]$
- R is the reward function. $R : S \times A \times S \to \mathbb{R}$

The function $R(s, a)$ defines the reward the agent receives for executing action a in state s. Let $\phi : S \times A \times S \to [0, 1]^n$ be a vector representation of state-action pairs, which is also known as the feature vector. The features are problem specific and are manually chosen, usually they are the observable quantities in the problem. We assume that the reward function is given by

$$R(s, a, s') = \sum_{i=1}^{m} \omega_i \phi_i(s, a, s') = \boldsymbol{\omega}^T \boldsymbol{\phi}(s, a, s') \qquad (1)$$

where $\omega_i \in \mathbb{R}$ defines the weight of the feature $\phi_i(s, a, s')$ in the feature vector $\phi(s, a, s')$. The values of the weights are constrained such that the L2 norm of the weights is unity. These weights are what we learn using Inverse Reinforcement learning.

A deterministic policy π defines a mapping from states to action and defines which action a is taken in a state s. A stochastic policy is a probability distribution over actions

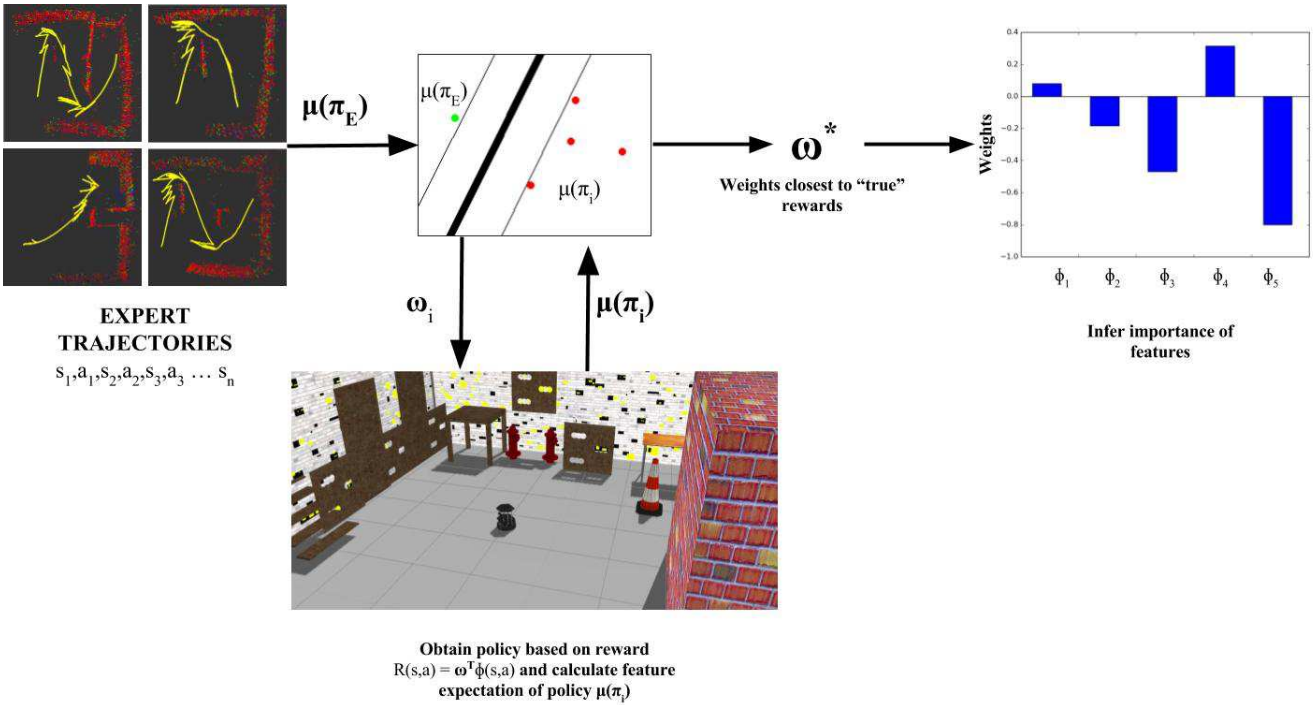

EXPERT
TRAJECTORIES
s1,a1,s2,a2,s3,a3 ... sn
μ(π_E)
μ(π_E)
μ(π_i)
ω_i
μ(π_i)
ω*
Weights closest to "true" rewards
Weights
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
φ1
φ2
φ3
φ4
φ5
Infer importance of features
Obtain policy based on reward
R(s,a) = ω^T φ(s,a) and calculate feature
expectation of policy μ(π_i)

Figure 1: Overview of using IRL to learn a reward function for Active Monocular SLAM

in a given state. The value function for a state, given a policy π, is a measure of the performance of the policy and is calculated as the sum of discounted future rewards and can be expressed as

$$\begin{aligned} V_{\omega}^{\pi}(s_0) &= \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) \\ &= \sum_{t=0}^{\infty} \gamma^t \boldsymbol{\omega}^T \boldsymbol{\phi}(s_t, a_t, s_{t+1}) \end{aligned} \tag{2}$$

Since $\boldsymbol{\omega}$ is independent of the time, we can take it outside the summation.

$$V_{\omega}^{\pi}(s_0) = \boldsymbol{\omega}^T [\sum_{t=0}^{\infty} \gamma^t \boldsymbol{\phi}(s_t, a_t, s_{t+1})] \tag{3}$$

The value function can now be thought of as a weighted sum of the future discounted feature values, or the **feature expectations**, denoted by $\mu(\pi)$ of a policy π is calculated as the sum of discounted feature values $\phi(s_t, a_t, s_{t+1})$

$$\boldsymbol{\mu}(\pi) = \sum_{t=0}^{\infty} \gamma^t \boldsymbol{\phi}(s_t, a_t, s_{t+1}) \tag{4}$$

Where $\mu(\pi)$ is a vector of length m equal to the number of individual features ϕ_i, and contains the individual feature counts $\mu_i(\pi)$. The feature expectations of a policy are independent of the weights of the reward function, they only depend on the states visited and the actions performed during the run. Also, the expected return of a policy π can then be written as,

$$V_{\omega}^{\pi}(s_0) = \sum_{i=1}^{m} \omega_i \mu_i(\pi) = \boldsymbol{\omega}^T \boldsymbol{\mu}(\pi) \tag{5}$$

Expert feature expectations $\mu(\pi_E)$ are calculated by observing the expert behavior and recording the visited states and the immediate rewards as we do with any other policy. Given the expert feature expectation $\mu(\pi_E)$, Inverse Reinforcement Learning tries to find weights such that the reward function corresponding to these weights resembles the underlying reward function the expert demonstrator is trying to maximize.

3. COMPUTATIONAL MODEL FOR IRL

This section talks about the IRL framework used in the paper and the specifications of our approach in solving the problem of avoiding Monocular SLAM failure. We first discuss the reward function features used in the context of the problem, then talk about the importance of reward function in the learning process, then introduce the expert used in our problem and finally talk about the Reinforcement learning model used to solve the MDP.

Figure 1 shows a flowchart of our method. We first observe the expert over multiple trajectories. Further details on this can be found in section 3.3. Given the set of expert trajectories, we find out what is the corresponding expert feature expectation by averaging over these trajectories. Once we have the expert feature expectation, we use Inverse Reinforcement Learning to learn an underlying reward function. IRL iteratively generates weights, obtained as a result of an optimization problem. Details regarding the implementation of IRL are discussed in section 3.2. Once a set of weights are obtained, we calculate our reward function as shown in equation (1) and form a policy by applying Reinforcement Learning with this reward function. We calculate the corresponding feature expectation of that policy, which is added as a constraint in the optimization problem and a new set of weights are then obtained. This process repeats until convergence. Once the IRL converges, examining the optimal weight vector can give us further insight as to how each feature affects Monocular SLAM and which features play a bigger role than others.

3.1 Reward Function Parameters

In order to have a robust model, our parameters need to be selected in such a way that our model remains map agnostic, eliminating the need to re-train our model in new environments. This requires us to ignore parameters that are specific to a given map or environment. The parameters need to be chosen such that they help capture information about the way the robot moves and the way Monocular SLAM reacts accordingly. More specifically, since we are trying to learn navigational strategies to avoid SLAM failure, we need to look closer at the parameters that play a role in causing SLAM to fail. Further insight as to why they would help us achieve our goal is discussed below.

Failure in Monocular SLAM systems usually occurs when we enter areas of low feature density. This causes a break in continuity of features tracked between subsequent views. The estimate of the trajectory comes from the way the features are tracked over multiple frames. Thus, feature overlap is an important aspect of calculating the pose estimate. It is highly desirable to have large amounts of feature overlap between frames as a lesser number features can lead to a high error in the pose estimate.

Large rotations without adequate translation also add to the deterioration of pose estimates. This is due to the fact that in some cases, large rotations can get mapped to a translatory motion, causing large deviation in the pose estimate. This is a problem of feature based Monocular SLAM algorithms. In such cases, the tracking quality decreases causing a SLAM failure.

When performing Monocular SLAM with a handheld camera, the user typically executes multiple sequences of subsequent forward and backward motions. This is to give the algorithm differing viewpoints from which similar parts of the scene can be viewed, thereby strengthening the quality of the map made and consequently, the estimate of the camera trajectory.

The state-action representation used for the problem is a tuple of observables in the environment. Keeping in mind the above points, the parameters that we have considered for our formulation are the direction of motion, angle change $\Delta\theta$ and the common features seen between subsequent views, which can be thought of as overlap in the field of view between subsequent views, denoted as ΔFOV.

One additional feature that we have considered is the condition of SLAM failure itself. Since the quality of SLAM changes after executing each action, this change, if crucial, needs to be accordingly penalized. In our case, since we are trying to prevent SLAM failures, we have a flag denoting whether SLAM has failed or not.

In our implementation, we restrict the motion of the robot to a maximum of 27°. While discretizing, we kept a cap of 600 features on the overlap of features between two views. While training, consecutive views for which parameters were calculated were at a distance of 1m on average.

3.2 Learning the reward function

The reward function defines the goal of the task and shapes the policy optimization process, thus it is a crucial part of the MDP. By manipulating the reward function we can expect very different solutions to the MDP as the learned behaviour is sensitive to the provided reward function. Usually, it is assumed that the reward function is given in an MDP, however it is hard to specify the reward function beforehand for solving a complex task. This problem of crafting the reward function is even more relevant when the task requires an intuitive understanding of the problem. This problem of designing the right reward functions led to the emergence of IRL methods under the umbrella of Learning from demonstration frameworks. Given the actions of an agent whose actions are assumed optimal for the problem and the sensory information of the agent, the goal of IRL is to come up with a reward function that can justify the demonstrated behaviour.

The algorithm that we follow for performing IRL and obtaining the weights is given in Algorithm 1. We first observe the expert trajectories and calculate the feature expectation, $\mu(\pi_E)$, as per equation (4) based on the parameters described in section 3.1. We do the same with a random policy to initialize our policy set. We obtain a set of weights by solving the optimization problem as stated in Algorithm 1, which would give us our new reward function as per equation (1). Once this is done, a policy is obtained using Reinforcement Learning for the new reward function, as described in section 3.4. The corresponding feature expectation of the policy is calculated and added in the constraints of the optimization problem.

Algorithm 1 Inverse RL

```
Approximate $\mu(\pi_E)$ from empirical data
Initialize $\mu(\pi_1)$ by following a random policy
Initialize $\omega_1$ randomly
$i \leftarrow 1$
Repeat :
  Obtain $\omega_{i+1}$ by solving the following optimization
      minimize      $||\omega||_2^2$
      subject to    $\omega^T\mu(\pi^E) >= 1$
                    $-\omega^T\mu(\pi^j) >= 1 \quad ; j = 1, \dots i$
  if $\omega_{i+1}^T(\mu(\pi^E) - \mu(\pi^i)) <= \epsilon$ or $||\omega_{i+1} - \omega_i||_2^2 <= \epsilon$
  then
      terminate
  else
      Learn Q-values corresponding to weights $\omega_{i+1}$
      Obtain $\pi^{i+1}$ corresponding to the learnt Q-values
      Obtain $\mu(\pi^{i+1})$
      $i \leftarrow i + 1$
```

Once the algorithm converges, we manually inspect the set of weights that have been calculated, and choose those weights which give the best performance over all the iterations.

3.3 Observing Expert Behaviour

The expert in this problem is not a human, rather it is an agent trained in a Reinforcement learning framework with a hand crafted reward function. This simple RL agent can learn the manoeuvres that lead to a SLAM-safe behaviour, thus we decided to employ this RL agent as the expert from whose behaviour we shall try to learn the task of avoiding actions that might lead to Monocular SLAM breakage.

The feature expectation of the expert $\mu(\pi^E)$ is calculated from the recorded state-action pairs observed during multiple episodes, from Monocular SLAM initialization till failure. Steps in each episode were taken while the robot is taking actions according to the expert policy π^E, which comes from a handcrafted reward function, which can be seen in Algorithm 2.

Algorithm 2 Calculation of Reward

```
procedure GetReward( $\Delta FOV$, $\Delta\theta$, $Status_{PTAM}$ )
    $Reward_{FOV} \leftarrow$ -10 + $max(600, \Delta FOV)/60$
    $Reward_{angle} \leftarrow$ -$|\Delta\theta|$/10
    if $Status_{PTAM}$ == $BROKEN$ then
        $Reward_{PTAM} \leftarrow$ -10
    else
        $Reward_{PTAM} \leftarrow$ 0
    return $Reward_{FOV} + Reward_{angle} + Reward_{PTAM}$
```

3.4 Reinforcement Learning Model

Reinforcement Learning [14] aims at learning how to maximize not just the immediate gains but also the sum of future discounted gains. This return is referred to as the Value of the state s, denoted by $V(s)$. In terms of actions, the effectiveness of performing an action a from a state s is given by the Q-value of that state action pair, denoted by $Q(s, a)$. Learning these value functions involves interaction with the system and updating the values and qualities based on the results of these interactions.

RL methods, such as Temporal Difference (TD) Learning, are used to learn the quality function through experience. Q-learning [19][20] is a widely used Temporal Difference (TD) prediction method for control problems. Q-learning returns an optimal set of Q-values, irrespective of the policy followed which implies that it is an off-policy learning method.We start with an initial set of Q-values which get update according the to Bellman equation, stated in equation (6). The values are updated incrementally after either episodes or samples, in a way similar to dynamic programming. The update for performing an action a from a state s, reaching a new state s' and obtaining a reward r is as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \quad (6)$$

where a' is the action that would be performed from the next state s', α, the learning rate and γ, the discount factor. Once the Q-values are learnt, the optimal policy would be to choose the action would be that which maximizes the Q-value from the current state.

As mentioned in Algorithm 1, we need to learn the Q-values corresponding to the obtained reward function, from which we derive a policy. Once the Q-values are learnt, we follow a 95% exploitation policy which is what we calculate our feature expectation for. Once this has been calculated, it gets added a constraint in the optimization problem for finding the weights and a new set of weights are obtained. The expert policy is also calculated similarly by following a 95% exploitation policy based on the expert's reward function.

In our implementation, we have used Q learning, as mentioned above, to learn the Q-values. For storing the Q-values, we use a lookup table representation for the Q function, by discretizing the states and actions as described in Section 3.1. Details of this discretization is given in table 1.

Table 1: Discretization of State-Action pairs

Parameter	Range	No. of values
dir	1 or 2	2
$\Delta\theta$	0° to 30°	20
ΔFOV	0 - 600	20

4. EXPERIMENTATION AND RESULTS

Gazebo [21] is a framework that accurately simulates robots and dynamic environments similar to ones that a robot would encounter in the real world. It also models the interactions between objects realistically. It allows for the easy creation of complex maps as well. Because of these factors, we decided to choose Gazebo as out training and testing platform. Experiments were performed in simulated environments as shown in Fig. 2 on a Turtlebot using a Microsoft Kinect for the RGB camera input.

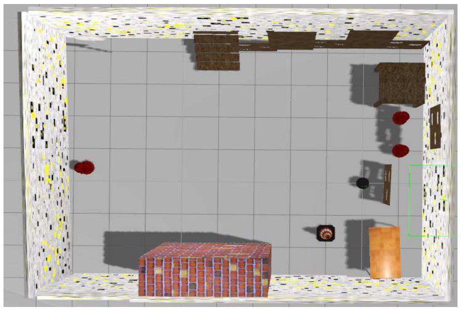


Figure 2: The training and testing are done in a simulated environment in Gazebo with a Kinect mounted on a Turtlebot

4.1 Training

For our experiments, we have decided to use PTAM to perform SLAM. We learn the Q-values offline between every IRL iteration using Q-learning, as described in Section 3.4. This makes the process of learning the policy faster as compared to running the agent with the Monocular SLAM in loop. For offline training, we learn the Q-values from around 13,500 samples (Q-matrix has 800 elements) of state-action pairs obtained with PTAM in loop while the agent is following a random policy. It is possible that some state-action pairs in the Q-matrix might not be visited enough number of times, thus we interpolate the matrix by performing a Stochastic Gradient Descent Regression, implemented using the scikit-learn toolbox [23], initialized with the immediate rewards of the respective state-action pair.

4.2 Testing

After learning the Q-values, we go on to obtain the feature expectation by making the agent follow an ϵ-greedy policy with 95% exploitation. This is done with the Monocular SLAM in loop. We run the policy for 20 episodes and average the feature expectation over all the episodes according to equation (4). An episode is defined as the steps executed by the robot from PTAM initialization till PTAM failure. Once a feature expectation is obtained, it is added to the set of constraints in the optimization problem described in Algorithm 1.

To verify the underlying truth of our reward function, we use two different criteria. The first is the effectiveness of the learnt model in avoiding SLAM failures. The second is usefulness of the model in navigating between start and goal locations in various maps, by while continuously evaluating the robot motion based on the learnt Q-values. The first can be seen in our testing phase by comparing the average number of steps taken in each episode with that of the hand-crafted reward function.

For the second, sets of experiments experiments were conducted on different maps in Gazebo (Fig. 4) by navigating a Turtlebot between start and goal locations, using waypoint navigation. During navigation, we constantly keep checking if the subsequent part of the trajectory that is being traversed would lead to a SLAM failure or not. This detection is done by checking the Q-value of an action against a threshold. If it is less than the threshold, we consider it as an unsafe action, otherwise it is a safe action and the robot continues on the path. Once an unsafe action is detected, we stop the robot and execute a safe action, from a set of candidate safe actions, which best aligns the robot with the global path it was initially traversing. Both the regular planner and SLAM-Safe planner use 5th order Bernstein curves for path planning [18].

The threshold is chosen such that it maximises the true predictions and minimizes the false predictions in the training data used to learn the Q-values. For our experiments, we found -0.2 to be a suitable threshold.

Experiments were carried out on a laptop with Intel Core i7-5500U 2.40GHz CPU running Ubuntu 14.04 using Robot Operating System (ROS) [22] for controlling the robot and performing SLAM.

5. RESULTS AND DISCUSSIONS

The IRL algorithm terminates after around 6-7 iterations on an average. The weights obtained are shown in table 2.

Table 2: weights obtained from the IRL algorithm

Features	Backward	Forward	$\Delta\theta$	ΔFOV	SLAM Failure
Weights	0.0801	-0.1831	-0.4698	0.3127	-0.8009

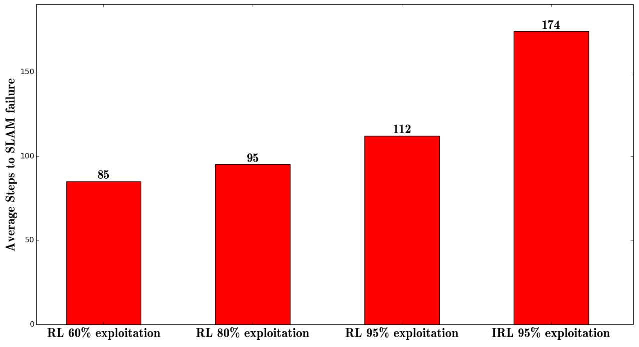
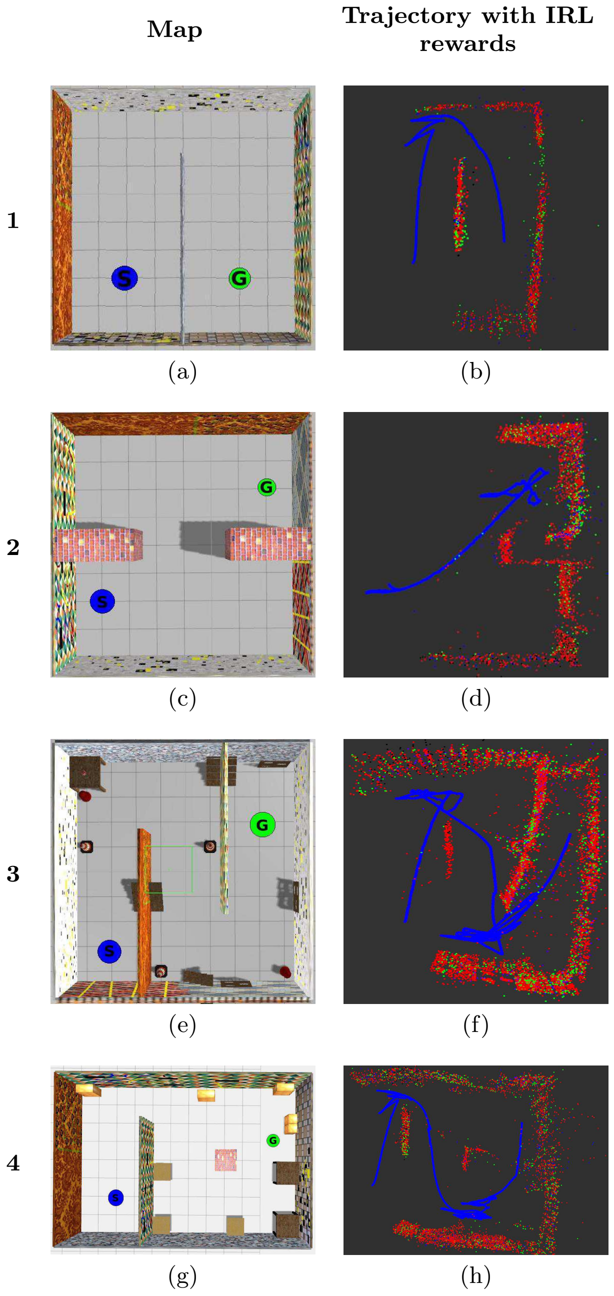


Figure 3: The average steps to taken by the robot until SLAM failure in the training phase.



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 4: Results of Goal based trajectory navigation in Gazebo. The first column shows the environment used for experiments. The second column is the Trajectory estimate and Map Point Cloud when using IRL to prevent failures.

The obtained weights are quite intuitive and capture what the expert is trying to optimize. Similarities can be seen between the way the expert has intuitively assigned penalties and how IRL has learnt the features to penalize and the features to reward.

The **backward motion** feature is assigned a positive weight, encouraging the agent to take reversals when required, which was a very important factor in the expert demonstrations as well. This stems from the fact that taking a backward step would widen the field of view of the agent resulting in viewing more area along with giving a high feature overlap.

The high weight for ΔFOV, the **feature overlap** between subsequent frames can be seen as a consequence of this. Low overlap of features between frames is one of the main causes for PTAM failure. A positive reward should be given for this feature indicating a higher reward for higher overlap of features.

The **forward motion** is assigned a small negative weight, which could possibly mean that taking too many forward going actions may lead to SLAM breakage and the algorithm tries to tell the agent to avoid them.

The algorithm also very intuitively assigns a high negative weight to changes in **angular orientation**, as it is obvious that a large $\Delta\theta$ would result in low correspondence between successive frames. Other than that, large angle changes are more prone to cause large deviations in the pose estimate, which can lead to failure much quicker.

Finally, the algorithm also learns to highly penalize steps that cause a SLAM failure which can be seen by the highly negative weight corresponding to PTAM failure. This draws from the fact that the expert has also given a large negative penalty for PTAM failure.

As we can clearly see in Fig 3, the agent trained with the IRL obtained reward function performs much better than than the agent trained with a hand-crafted reward function. We also report the performances of a new map environment and record the number of successful and unsuccessful attempts by each algorithm in terms of reaching the goal safely without SLAM breakage.

Table 3: Results for Goal Based Trajectory Planning

Map	Planner Type	Runs	Success	Failures	Success %
1	RL	10	9	1	90
1	IRL	10	10	0	100
2	RL	10	8	2	80
2	IRL	10	9	1	90
3	RL	10	8	2	80
3	IRL	10	7	3	70
4	RL	10	6	4	60
4	IRL	10	8	2	80

Table 3 summarizes the results of our goal based navigation experiments which can be seen qualitatively in Fig. 4.The first column in Fig. 4 shows the four simulation environments. The start and goal locations are shown in these environments as blue and green circles, marked with the letters "S" and "G" respectively. The second column shows failure runs due to a state-of-the-art planner as the robot fails to reach the goal due to SLAM breakage. The third column shows the trajectories executed by the SLAM-Safe planner learnt using the IRL rewards.

Interestingly, the trajectories generated by the IRL reward function in Fig. 4 show significant number of motions in reverse near regions of low feature continuity. This is similar to motions seen while using the expert, which shows the effectiveness of IRL in learning the behaviour exhibited by the expert.

Another noteworthy aspect of the current approach is the ability of the IRL formulation (called IRL agent henceforth) to improvise and enhance the expert policy learned by the RL formulation/RL agent. This is showcased in various ways in Table 3 and in Fig. 3. In other words this paper reinforces the notion that a policy resulting from a handcrafted reward scheme based on human expertise, even if such a policy is learned in a principled way through an RL framework, can further be enhanced significantly through the proposed framework. Thus the IRL agent is able to learn possibly a different reward function, which is even better than the human intuited reward.

6. CONCLUSION

Automating Monocular SLAM or Active Localization for SLAM has been a significantly challenging problem to solve due to the idiosyncrasies of a monocular camera. Monocicular SLAM breakages and outages are common over a run even if the camera is carefully moved or teleoperated by an expert. This paper proposes a novel data driven strategy for learning handcrafted expert behavior. The proposed IRL strategy learns such expert intuited policies and outperforms the expert in various maps through enhanced SLAM longevities and repeatable goal reaching behavior for diverse locations on a variety of maps. The immediate scope of this effort lies in integrating the proposed framework within a monocular SLAM based exploration system.

REFERENCES

[1] Ng, Andrew Y., and Stuart J. Russell. "Algorithms for inverse reinforcement learning." Icml. 2000.

[2] Abbeel, Pieter, and Andrew Y. Ng. "Apprenticeship learning via inverse reinforcement learning." Proceedings of the twenty-first international conference on Machine learning. ACM, 2004.

[3] Ratliff, Nathan D., J. Andrew Bagnell, and Martin A. Zinkevich. "Maximum margin planning." Proceedings of the 23rd international conference on Machine learning. ACM, 2006.

[4] Ramachandran D, Amir E (2007) Bayesian inverse reinforcement learning. In: Proceedings of the 20th International Joint Conference of Artificial Intelligence (IJCAI), pp 2586âĂŞ2591

[5] Zhifei, Shao, and Er Meng Joo. "A survey of inverse reinforcement learning techniques." International Journal of Intelligent Computing and Cybernetics 5.3 (2012): 293-311.

[6] Leung, Cindy, Shoudong Huang, and Gamini Dissanayake. "Active SLAM using Model Predictive Control and Attractor Based Exploration." *In Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, pp. 5026-5031. IEEE, 2006.*

[7] Leung, Cindy, Shoudong Huang, Ngai Kwok, and Gamini Dissanayake. "Planning under Uncertainty using Model Predictive Control for Information Gathering." *Robotics and Autonomous Systems 54, no. 11 (2006): 898-910.* Harvard

[8] Kollar, Thomas, and Nicholas Roy. "Efficient Optimization of Information-Theoretic Exploration in SLAM." *In AAAI, vol. 8, pp. 1369-1375. 2008.*

[9] Indelman, Vadim, Luca Carlone, and Frank Dellaert. "Planning in the Continuous Domain: A Generalized Belief Space Approach for Autonomous Navigation in Unknown Environments." *The International Journal of*

Robotics Research 34, no. 7 (2015): 849-882.

[10] Charrow, Benjamin, Gregory Kahn, Sachin Patil, Sikang Liu, Ken Goldberg, Pieter Abbeel, Nathan Michael, and Vijay Kumar. ”Information-Theoretic Planning with Trajectory Optimization for Dense 3D Mapping.” *Proceedings of the Robotics: Science and System (RSS) (2015).*

[11] Kaess, Michael, Ananth Ranganathan, and Frank Dellaert. ”iSAM: Incremental Smoothing and Mapping.” *Robotics, IEEE Transactions on 24, no. 6 (2008): 1365-1378.*

[12] Mostegel, Christian, Andreas Wendel, and Horst Bischof. ”Active Monocular localization: towards autonomous Monocular exploration for multirotor MAVs.” 2014 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2014.

[13] Prasad, Vignesh, et al. ”SLAM-Safe Planner: Preventing Monocular SLAM Failure using Reinforcement Learning.” arXiv preprint arXiv:1607.07558 (2016).

[14] R.S. Sutton and A.G. Barto, ”Reinforcement Learning: An Introduction”, *MIT Press, 1998.*

[15] Klein, Georg, and David Murray. ”Parallel Tracking and Mapping for Small AR Workspaces.” *In Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on, pp. 225-234. IEEE, 2007.*

[16] Mur-Artal, Raul, J. M. M. Montiel, and Juan D. Tardos. ”ORB-SLAM: a versatile and accurate Monocular SLAM system.” Robotics, IEEE Transactions on 31, no. 5 (2015): 1147-1163.

[17] Engel, Jakob, Thomas SchÃűps, and Daniel Cremers. ”LSD-SLAM: Large-scale direct Monocular SLAM.” In Computer VisionâĂŞECCV 2014, pp. 834-849. Springer International Publishing, 2014.

[18] Gopalakrishnan, Balasubramanian, Arun Kumar Singh, and K. Madhava Krishna. ”Time Scaled Collision Cone Based Trajectory Optimization Approach for Reactive Planning in Dynamic Environments.” *In Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on, pp. 4169-4176. IEEE, 2014.*

[19] Watkins, Christopher John Cornish Hellaby. ”Learning from delayed rewards.”, *PhD diss., University of Cambridge, 1989.*

[20] Watkins, Christopher JCH, and Peter Dayan. ”Q-learning.”, *Machine learning 8.3-4 (1992): 279-292.*

[21] Koenig, Nathan, and Andrew Howard. ”Design and use paradigms for gazebo, an open-source multi-robot simulator.” Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on. Vol. 3. IEEE, 2004.

[22] Quigley, Morgan, et al. ”ROS: an open-source Robot Operating System.” ICRA workshop on open source software. Vol. 3. No. 3.2. 2009.

[23] Pedregosa, Fabian, et al. ”Scikit-learn: Machine learning in Python.” Journal of Machine Learning Research 12.Oct (2011): 2825-2830.