Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика, искусственный интеллект и системы управления

КАФЕДРА Системы обработки информации и управления

Домашнее задание
по дисциплине «Методы машинного обучения»

Выполнил: Богданов Д.А.

Группа: ИУ5-24М

Проверил: Гапанюк Ю.Е.

Москва, 2022 г.

ОГЛАВЛЕНИЕ

1. ЗАДАНИЕ

Домашнее задание по дисциплине направлено на анализ современных методов машинного обучения и их применение для решения практических задач. Домашнее задание включает три основных этапа:

1. выбор задачи;
2. теоретический этап;
3. практический этап.

Этап выбора задачи предполагает анализ ресурса paperswithcode [1]. Данный ресурс включает описание нескольких тысяч современных задач в области машинного обучения. Каждое описание задачи содержит ссылки на наиболее современные и актуальные научные статьи, предназначенные для решения задачи (список статей регулярно обновляется авторами ресурса). Каждое описание статьи содержит ссылку на репозиторий с открытым исходным кодом, реализующим представленные в статье эксперименты. На этапе выбора задачи обучающийся выбирает одну из задач машинного обучения, описание которой содержит ссылки на статьи и репозитории с исходным кодом.

Теоретический этап включает проработку как минимум двух статей, относящихся к выбранной задаче. Результаты проработки обучающийся излагает в теоретической части отчета по домашнему заданию, которая может включать:

- описание общих подходов к решению задачи;
- конкретные топологии нейронных сетей, нейросетевых ансамблей или других моделей машинного обучения, предназначенных для решения задачи;
- математическое описание, алгоритмы функционирования, особенности обучения используемых для решения задачи нейронных сетей, нейросетевых ансамблей или других моделей машинного обучения;
- описание наборов данных, используемых для обучения моделей;
- оценка качества решения задачи, описание метрик качества и их значений;
- предложения обучающегося по улучшению качества решения задачи.

Практический этап включает повторение экспериментов авторов статей на основе представленных авторами репозиториев с исходным кодом и возможное улучшение обучающимися полученных результатов. Результаты проработки обучающийся излагает в практической части отчета по домашнему заданию, которая может включать:

- исходные коды программ, представленные авторами статей, результаты документирования программ обучающимися с использованием диаграмм UML, путем визуализации топологий нейронных сетей и другими способами;
- результаты выполнения программ, вычисление значений для описанных в статьях метрик качества, выводы обучающегося о воспроизводимости экспериментов авторов статей и соответствии практических экспериментов теоретическим материалам статей;
- предложения обучающегося по возможным улучшениям решения задачи, результаты практических экспериментов (исходные коды, документация) по возможному улучшению решения задачи.

Отчет по домашнему заданию должен содержать:

1. Титульный лист.
2. Постановку выбранной задачи машинного обучения, соответствующую этапу выбора задачи.
3. Теоретическую часть отчета.
4. Практическую часть отчета.
5. Выводы обучающегося по результатам выполнения теоретической и практической частей.
6. Список использованных источников.

2. ПОСТАНОВКА ЗАДАЧИ

В результате анализа содержимого ресурса «Papers with code» была выбрана область навигации робота (**Robot Navigation).** Фундаментальная цель мобильной робототехники — достичь цели без столкновений. Мобильный робот должен знать о препятствиях и свободно перемещаться в различных рабочих сценариях.

В данной области было решено изучить решения задачи навигации по точкам и целям (PointGoal Navigation).

По проблеме навигации опубликовано значительное количество литературы. Однако большинство этих исследований сосредоточено на задаче навигации в известных (уже нанесенных на карту) средах. Несколько исследователей продемонстрировали высококачественные результаты при PointGoal навигации в неизвестной среде и в исследовательской задаче, где они использовали GPS в качестве входных данных.

Рисунок 1 демонстрирует страницу, посвященную данной задаче на ресурсе Papers With Code. После формального описания задачи (для данной задачи описание отсутствует) следует таблица метрик сравнения качества работы методов, предназначенных для решения поставленной задачи (Benchmarks).

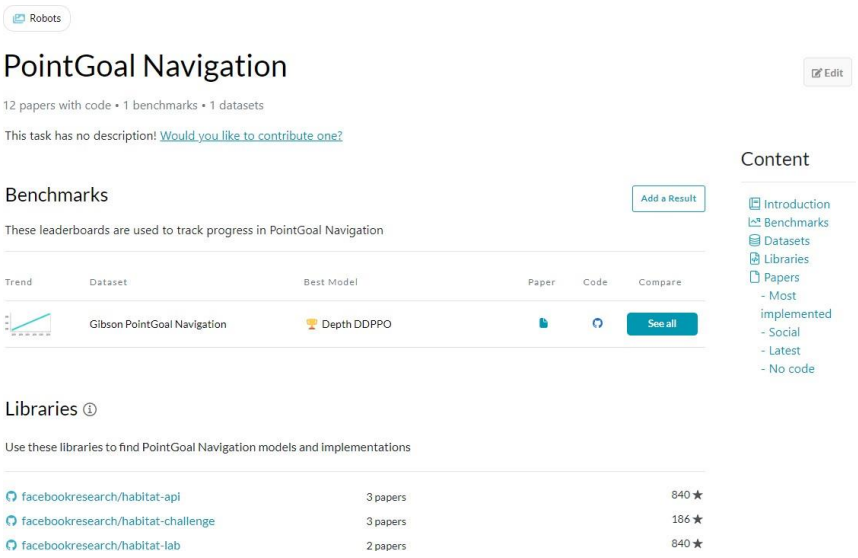

Рисунок 1 - Страница, посвященная PointGoal Navigation

Таблица состоит из колонок:

- Trend – тенденция качества решения задачи распознавания на соответствующем датасете;
- Dataset – набор данных, на котором производится оценка качества работы алгоритма распознавания;

- Best Model – метод (или модель машинного обучения), показавший на данный момент лучшие результаты в решении задачи распознавания на соответствующем датасете;
- Paper – индикатора наличия статьи, описывающей соответствующий метод;
- Code – индикатор наличия кода, реализующего соответствующий метод (модель);
- Compare – ссылка для перехода на новую страницу со сравнением всех моделей и датасетов.

Ниже можно найти следующие раздел, посвященный библиотекам (Libraries), содержащим проверенные временем модели, предназначенные для решения задачи навигации.

На той же странице, но ниже можно увидеть ещё два раздела(см. Рисунок 2) :

- раздел Datasets, содержащий все датасеты, которые можно использовать для решения поставленной задачи;
- раздел Most Implemented Papers, содержащий статьи, отсортированные по применяемости сопутствующим им кода и/или теоретической информации.

Последний раздел также можно отсортировать по недавним статьям, выпущенным по данной теме.

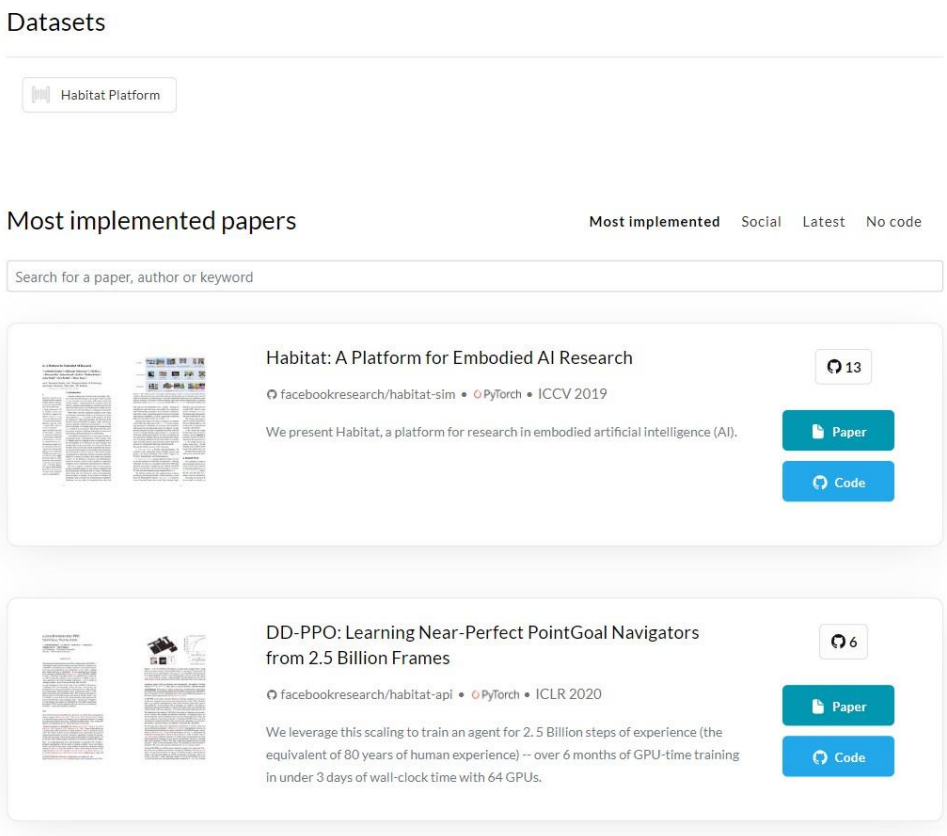

Рисунок 2 - Продолжение страницы, посвященной PointGoal Navigation

В рамках домашнего задание было интересно найти реализацию, использующую SLAM алгоритмы, что привело меня к статье «Learning to Explore using Active Neural SLAM» [2], которая представляет модульный и иерархический подход к обучению политик для исследования 3D окружения.

Рисунок 3 демонстрирует страницу, посвященную данной статье. На данной странице можно:

- прочитать Аннотацию (Abstract) к статье;
- получить текст статьи в формате PDF;
- открыть код, реализующий методы, описанные в данной статье;
- посмотреть задачи, решение которых предлагают авторы статьи;
- посмотреть наборы данных, на которых обучалась или тестировалась модель;
- посмотреть результаты решения задачи (необязательно указываются на данном сайте авторами статей)
- посмотреть методы, используемые авторами статьи для решения данной задачи.

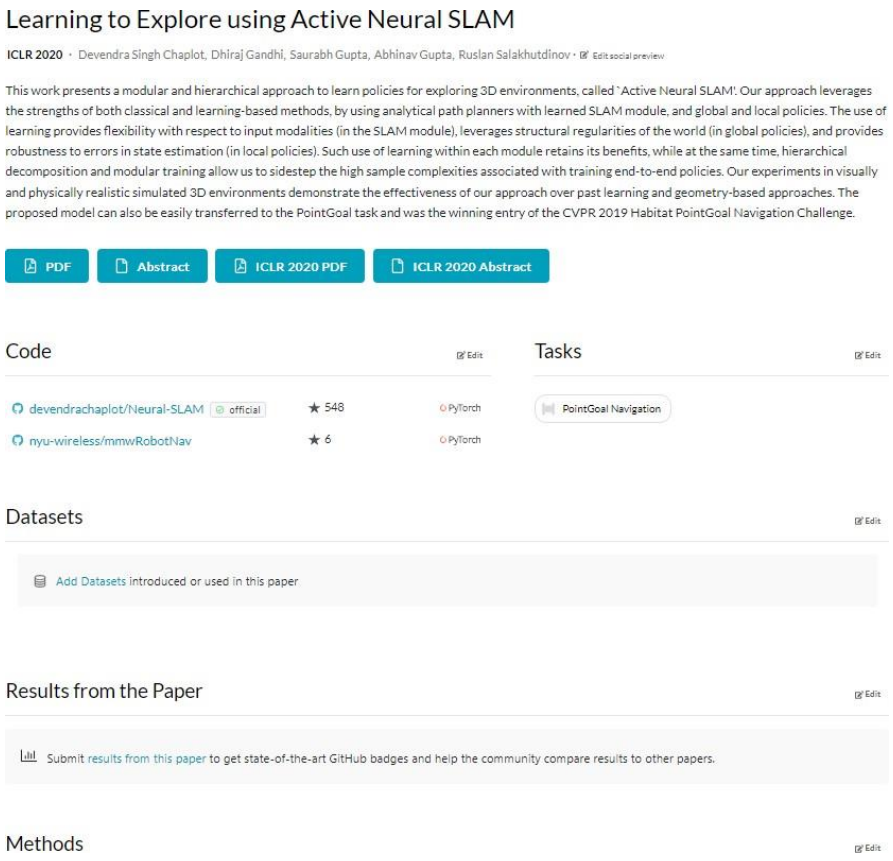

Рисунок 3 - Страница, посвященная статье

3. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

В рамках домашнего здания рассматриваются материалы статьи «Обучение исследованию с помощью активного нейронного SLAM» (Learning to Explore using Active Neural SLAM).

Навигация является критически важной задачей при создании интеллектуальных агентов. Навигационные задачи могут быть выражены в различных формах, например, задачи с точечными целями включают в себя навигацию по определенным координатам, а семантическая навигация включает в себя поиск пути к определенной сцене или объекту. Независимо от задачи, основная цель для навигации в неизвестной среде является исследование, т.е. как эффективно посетить как можно большую часть окружающей среды. Это полезно для максимизации охвата, чтобы дать наилучшие шансы на нахождение цель в неизвестной среде или для эффективного предварительного картирования среды при ограниченном бюджет времени.

В недавней работе Chen et al. (2019) для решения этой проблемы используется сквозное обучение, что объясняется следующим:

а) обучение обеспечивает гибкость в выборе входных модальностей (классические системы полагаются на наблюдение геометрии с помощью специализированных датчиков, в то время как обучающиеся системы могут определять геометрию непосредственно по RGB изображениям);

б) обучения может улучшить устойчивость к ошибкам в явной оценки состояния;

в) обучение может эффективно использовать структурные закономерности реального мира, что приводит к более эффективному поведению в ранее невидимых средах.

Это привело к разработке сквозной обучаемой политики на основе нейронной сети, которая обрабатывает необработанные сенсорные наблюдения для непосредственного вывода действий, которые должен выполнить агент.

Авторы демонстрируют предложенный подход в визуально и физически реалистичных симуляторах для задачи геометрического исследования (посещение как можно большей территории). Работа производилась с помощью симулятора Habitat от Savva et al. (2019). Хотя Habitat уже является визуально реалистичным (в нем используются сканы реального мира от Changet al. (2017) и Xia et al. (2018) в качестве окружения), авторы улучшают его физическую реалистичность, используя модели шума датчиков приведения в действие и модели шума датчиков одометрии, которые были собрана при проведении физических экспериментов на реальном мобильном роботе.

Авторы выкладывают наборы данных, код и обученные модели в общем доступе. Также существует веб-сайт проекта [3].

3.1. Описание работ, которые были использованы в рассматриваемой статье.

Навигация была хорошо изучена в классической робототехнике. В последнее время возродился интерес к использованию обучения для выработки навигационной политики для различных задач. Рассмотренная работа опирается на концепции классической робототехники и обучения для навигации. Ниже приводится обзор соответствующих работ.

Подходы к навигации. Классические подходы к навигации разбивают проблему на две части: картирование и планирование пути. Сопоставление осуществляется посредством одновременной локализации и сопоставления (Thrun et al., 2005; Hartley and Zisserman, 2003; Fuentes-Pacheco et al., 2015), путем объединения информации из нескольких видов окружающей среды. Хотя разреженная реконструкция может быть хорошо выполнена с монокулярными RGB-изображений (Mur-Artal and Tardós, 2017), плотное отображение неэффективно (Newcombe et al., 2011) или требует специализированных сканеров, таких как Kinect (Izadi et al., 2011). Карты используются для расчета пути к цели с помощью планирования пути (Kavraki et al., 1996; Lavalle and Kuffner Jr, 2000; Canny, 1988). Эти классические методы послужили источником вдохновения для современных методов, основанных на обучении. Исследователи разработали нейросетевые политики, которые рассуждают с помощью пространственных представлений (Gupta et al., 2017;Parisotto and Salakhutdinov, 2018; Zhang et al., 2017; Henriques and Vedaldi, 2018; Gordon et al,2018), топологических представлений (Савинов и др., 2018; 2019), или использовать дифференцируемые и обучаемые планировщики (Tamar et al., 2016; Lee et al., 2018; Gupta et al., 2017; Khan et al., 2017). Рассмотренная работа развивает эти исследования, и мы изучаем иерархическую и модульную декомпозицию проблемы и используем обучение внутри этих компонентов вместо сквозного обучения. Исследования также сосредоточены на включении семантики в SLAM (Pronobis and Jensfelt, 2012; Walter et al., 2013).

Исследование в навигации. В то время как ряд работ посвящен пассивному построению карт, планированию пути и обучению политике на основе целей, гораздо меньшее количество работ посвящено проблеме активной навигации.

SLAM, т.е. как активно управлять камерой для построения карты. Мы отсылаем читателей к FuentesPacheco et al. (2015) за подробным обзором, а ниже приводим краткое описание основных тем. В большинстве подобных работ представляют эту задачу как

частично наблюдаемый марковский процесс принятия решений (POMDP), который приблизительно решены (Martinez-Cantin et al., 2009; Kollar and Roy, 2008), и или стремятся найти последовательность действий, которая минимизирует неопределенность карт (Stachniss et al., 2005; Carlone et al., 2014).

Другое направление работы исследует, выбирая точки обзора (например, на границе между исследованными и неисследованными регионами (Dornhege and Kleiner 2013; Holz et al.) и неизученными регионами (Dornhege and Kleiner, 2013; Holz et al., 2010; Yamauchi, 1997; Xu et al,2017)). Последние работы Chen et al. (2019); Savinov et al. (2019); Fang et al. (2019) решают эту проблему с помощью обучения. Предлагаемые в работе модульные политики объединяют два последних направления исследований, и демонстрируют улучшения по сравнению с репрезентативными методами из этих двух направлений. Исследование также изучалась в более общем плане в RL в контексте компромисса между разведкой и эксплуатацией (Sutton and Barto, 2018; Kearns and Singh, 2002; Auer, 2002; Jaksch et al., 2010).

Иерархические и модульные политики. Иерархические РЛ (Dayan and Hinton, 1993; Sutton et al, 1999; Barto and Mahadevan, 2003) является активной областью исследований, направленных на автоматическое обнаружение иерархий для ускорения обучения. Однако это оказалось сложной задачей, и поэтому в большинстве работ прибегают к использованию иерархий, определяемых вручную. Например, в контексте навигации, Bansal et al.(2019) и Kaufmann et al. (2019) разрабатывают модульные политики для навигации, которые сопрягают выученные политики с низкоуровневыми регуляторами обратной связи. Иерархические и модульные политики также использовались для воплощенных ответов на вопросы (Das et al., 2018a; Gordon et al., 2018; Das et al., 2018b).

3.2. Задача

Авторы статьи следуют постановке задачи разведки, предложенной Chen et al. (2019), где целью является максимизация охвата при фиксированном бюджете времени. Охват определяется как общая площадь карты которую можно пройти. Цель - обучить политику, которая принимает наблюдение s_t на каждом временном отрезке t выдает навигационное действие a_t для максимизации покрытия.

Модель Active Neural SLAM состоит из трех модулей: глобальной политики, локальной политики и модуля Neural SLAM. Как показано ниже, модуль Neural-SLAM прогнозирует карту и оценку положения агента на основе входящих наблюдений RGB и показаний датчиков. Эта карта и поза используются глобальной политикой для вывода долгосрочной цели, которая преобразуется в краткосрочную цель с помощью

планировщика аналитического пути. Локальная политика обучена двигаться к этой краткосрочной цели. На рисунке 4 приведена схема рассматриваемого подхода.

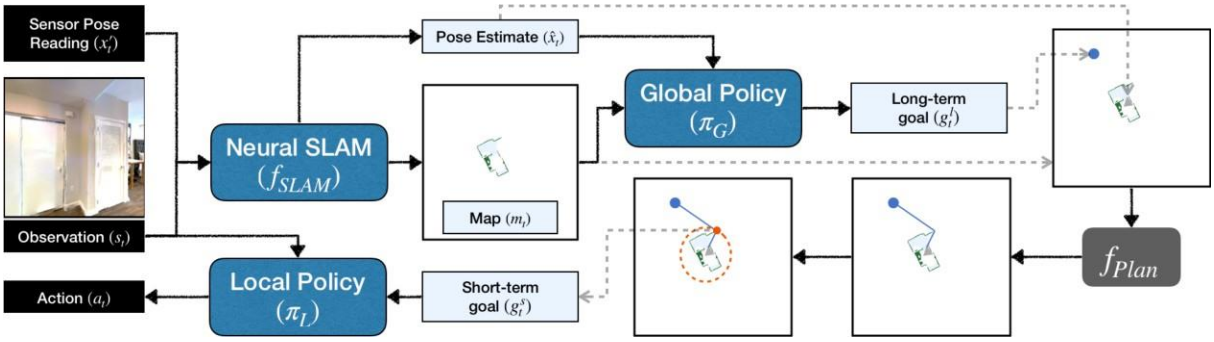

Рисунок 4 – Рассматриваемый подход.

3.3. Экспериментальная часть

Авторы статьи используют симулятор Habitat (Savva et al., 2019) с наборами данных Gibson (Xia et al., 2018) и Matterport (MP3D) (Chang et al., 2017) для экспериментов. И Gibson, и MP3D состоят из сцен, которые являются 3D-реконструкциями реального окружения, однако Gibson собран с использованием другого набора данных с камер, состоящего в основном из офисных помещений, в то время как MP3D состоит в основном из домов с большей средней площадью сцены. В статье был использован Gibson в качестве обучающего домена, а MP3D - для экспериментов по обобщению домена. Пространство наблюдения состоит из RGB-изображений размером 3 × 128 × 128 и показаний базовых датчиков одометрии размером 3 × 1, обозначающих изменение координат агента x-y и ориентации. Пространство действий состоит из трех действий: движение_вперед, поворот_влево, поворот_вправо. Как базовые показания датчиков одометрии, так и движение агента на основе действий являются зашумленными. Они реализованы с использованием моделей шума датчиков и действий, основанных на реальных данных.

Авторы следуют постановке задачи разведки, предложенной Chen et al. (2019), где цель максимизировать охват за фиксированный бюджет времени. Охват - это общая площадь карты, которую можно пройти. Авторы определяют проходимую точку как известную, если она находится в поле зрения агента и на расстоянии менее 3,2 м. от него. Мы используем две метрики оценки, абсолютную площадь покрытия в м2 (Cov) и процент (% Cov), т.е. отношение площади покрытия к максимально возможному покрытию в соответствующей сцене. Во время обучения каждый эпизод длится фиксированную длину в 1000 шагов.

Для обоих наборов данных использовалось разбиение на train/val/test, предоставленное Savva et al. (2019). Обратите внимание, что набор сцен, используемых в каждом разбиении, не совпадает, что означает, что агент тестируется на новых сценах, которые никогда не были видны во время обучения. Набор тестов Gibson не является публичным, а хранится на сервере онлайн-оценки для задачи Pointgoal. задачи Pointgoal. Мы используем эту оценку в качестве тестового набора для сравнения и анализа для домена Gibson. Валидационный набор используется для настройки гиперпараметров. Для анализа производительности всех моделей в зависимости от размера сцены, авторы разделили набор для проверки Гибсона на две части: небольшой набор из 10 сцен с исследованием. небольшой набор из 10 сцен с исследуемой площадью от 16 м2 до 36м2 и большой набор из 4 сцен с исследуемой площадью от 55 м2 до 100 м2. Обратите внимание, что размер карты обычно намного размер карты обычно намного больше, чем обследуемая площадь, причем самая большая карта имеет длину около 23 м и ширину 11 м.

Детали обучения. Мы обучаем нашу модель в домене Gibson и переносим ее в домен Matterport.
Картограф обучается предсказывать эгоцентрические проекции, а оценщик позы обучается предсказывать позы агента с помощью контролируемого обучения. Наземная истинная эгоцентрическая проекция вычисляется с использованием геометрических проекций на глубину. Глобальная политика обучается с помощью метода подкрепления обучения с вознаграждением, пропорциональным увеличению охвата в качестве награды. Локальная политика обучается с помощью имитационного обучения (клонирование поведения). Все модули обучаются одновременно. Их параметры независимы, но распределение данных взаимозависимо. На основе действий предпринятых локальной политикой, изменяется будущий входной сигнал для модуля Neural SLAM, который, в свою очередь, изменяет
карту и позу агента, вводимые в Глобальную политику, и, следовательно, влияет на краткосрочную цель, поставленную перед локальной политикой.
Базовые показатели. Мы используем ряд сквозных методов обучения с усилением (RL) в качестве базовых:
RL + 3LConv: RL Policy с 3-слойной конволюционной сетью, за которой следует GRU (Cho et al., 2014), как описано в Savva et al. (2019).
RL + Res18: политика RL, инициализированная с ResNet18 (He et al., 2016), предварительно обученной на ImageNet после чего используется GRU.
RL + Res18 + AuxDepth: Этот базовый уровень адаптирован из работы Mirowski et al. (2017), в которой в качестве вспомогательной задачи используется прогнозирование глубины предсказание глубины в качестве вспомогательной задачи. Мы используем ту же архитектуру, что и наш модуль Neural SLAM (conv слои из ResNet18) с одним дополнительным деконволюционным слоем для предсказания глубины, а затем 3 слоев свертки и GRU для политики.
RL + Res18 + ProjDepth: Этот базовый уровень адаптирован в работе Chen et al. (2019), которая проецирует глубину изображение в эгоцентрическом направлении сверху вниз в дополнение к RGB-изображению в качестве входных данных для политики RL. Поскольку у нас не имеем глубины в качестве входных данных, мы используем архитектуру из RL + Res18 + AuxDepth для предсказания глубины и проецируем предсказанную глубину перед передачей в 3Layer Conv и политику GRU.
Для всех базовых версий мы также передаем 32-мерную вставку показаний датчика позы в GRU вместе с представлением на основе изображения. Это вложение также изучается из конца в конец с помощью RL. Все базовые линии обучаются с помощью PPO (Schulman et al., 2017) с увеличением охвата по мере увеличения вознаграждения (идентично вознаграждению, используемому для обучения).
(идентично вознаграждению, используемому для Глобальной политики). Все базовые программы требуют доступа к во время обучения для вычисления вознаграждения. Наблюдение для глобальной политики, локальной политики и картографа также могут быть получены из карты "земля-истина". Оценщик позы требует дополнительного контроля в виде наземной истинной позы агента. Мы изучаем влияние этого дополнительного контроля в экспериментах по абляции.

	Gibson Val		Domain Generalization MP3D Test	
Method	% Cov.	Cov. (m2)	% Cov.	Cov. (m2)
RL + 3LConv [1]	0.737	22.838	0.332	47.758
RL + Res18	0.747	23.188	0.341	49.175
RL + Res18 + AuxDepth [2]	0.779	24.467	0.356	51.959
RL + Res18 + ProjDepth [3]	0.789	24.863	0.378	54.775
Active Neural SLAM (ANS)	**0.948**	**32.701**	**0.521**	**73.281**

Рисунок 5 - Эксплуатационные характеристики предложенной модели, Active Neural SLAM (ANS) и базовых моделей.

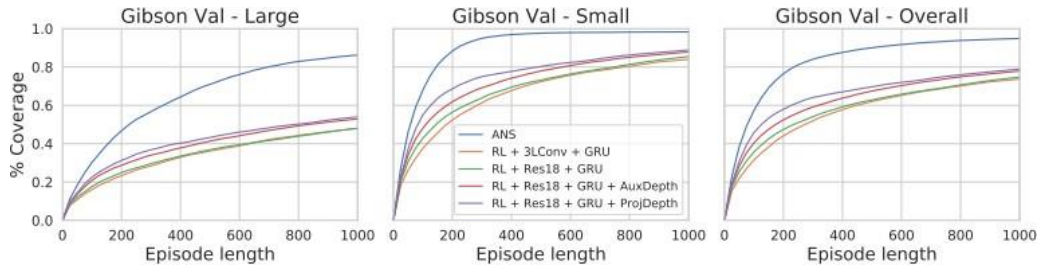

Рисунок 6 - График, показывающий % покрытия по ходу эпизода для ANS и базовых линий на больших и малых сцен в наборе Гибсон Вал, а также в общем наборе Гибсон Вал

3.4. Результат

Мы обучаем предложенную модель ANS и все базовые модели для задачи Exploration на 10 млн. кадров на обучающем наборе Gibson. Результаты показаны на рисунке 5 на наборе Gibson Val усреднены по 994 эпизодам в 14 различных невидимых сценах. Предложенная модель достигает среднего абсолютного и относительного охвата 32,701 м2/0,948 по сравнению с 24,863 м2/0,789 для по сравнению с 24,863м2/0,789 для лучшей базовой модели. Это указывает на то, что предложенная модель является более эффективной и действенной при исчерпывающем разведки по сравнению с базовой моделью. Это объясняется тем, что новая иерархическая архитектура политики уменьшает горизонт проблемы долгосрочной разведки, так как вместо того, чтобы предпринимать десятки низкоуровневых навигационных действий, глобальная политика принимает только несколько действий для достижения долгосрочной цели. Все модели, обученные на Gibson, оцениваются на домене Matterport. ANS приводит к более высокой производительности обобщения домена (73,281 м2/0,521 против 54,775 м2/0,378). Абсолютное покрытие выше % Cov ниже для домена Matterport, поскольку он состоит в среднем из более крупных сцен. На наборе небольших тестовых сцен MP3D (сравнимых с размерами сцен Gibson), ANS достиг производительности 31,407 м2/0,836 по сравнению с 23,091 м2/0,620 для лучшей базовой линии. Некоторые визуализации выполнения политики представлены на рисунке 7.

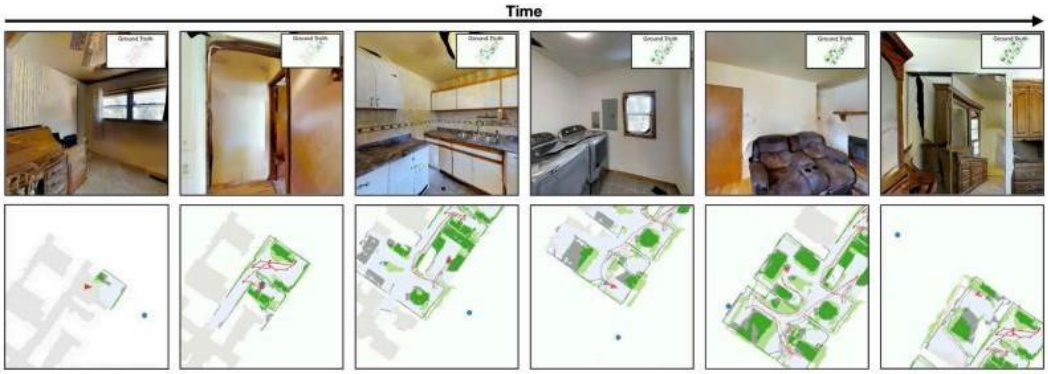

Рисунок 7 – Визуализация разведки

4. ПРАКТИЧЕСКАЯ ЧАСТЬ

Чтобы облегчить понимание статьи авторы выложили все материалы в репозиторий[4].

4.1. Обзор страницы в GitHub

Рисунок 8 демонстрирует страницу на сайте GitHub, посвященную данной статье.

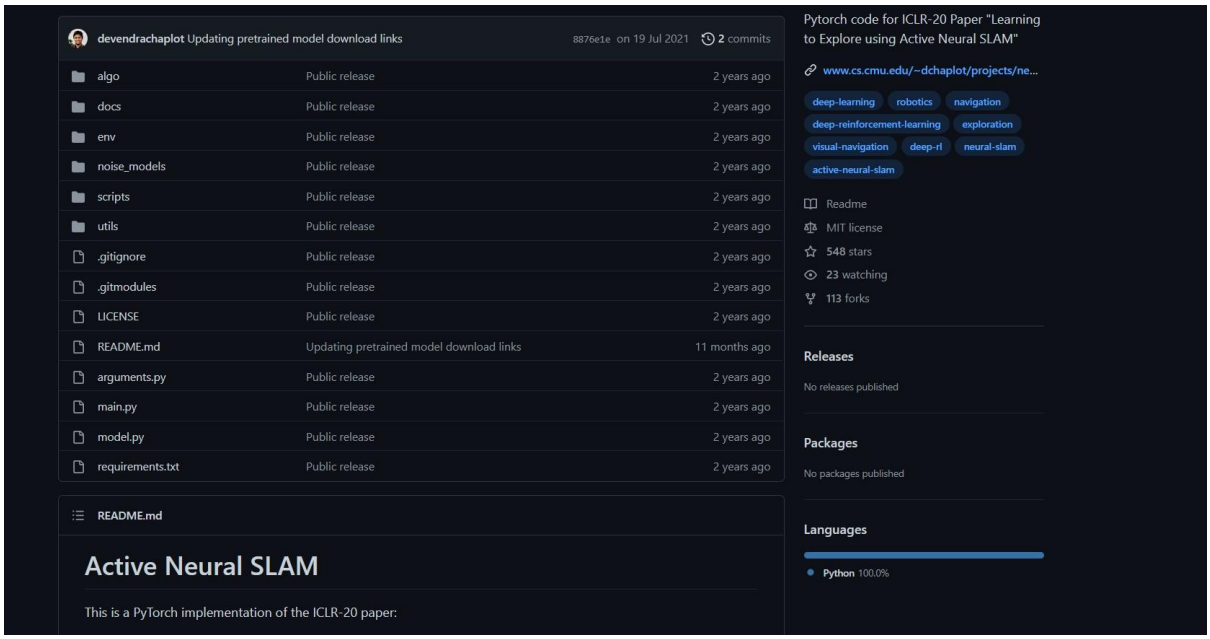

Рисунок 8 - Страница статьи в GitHub

На данной странице авторы дают краткое описание содержимого репозитория, а также описывают порядок работы с ним в нескольких разделах страницы.

4.2. код рассмотренной модели

```
import time
from collections import deque

import os

os.environ["OMP_NUM_THREADS"] = "1"
import numpy as np
import torch
import torch.nn as nn
from torch.nn import functional as F

import gym
import logging
from arguments import get_args
from env import make_vec_envs
from utils.storage import GlobalRolloutStorage, FIFOMemory
from utils.optimization import get_optimizer
```

```
from model import RL_Policy, Local_IL_Policy,
Neural_SLAM_Module

import algo

import sys
import matplotlib

if sys.platform == 'darwin':
    matplotlib.use("tkagg")
import matplotlib.pyplot as plt

# plt.ion()
# fig, ax = plt.subplots(1,4, figsize=(10, 2.5),
facecolor="whitesmoke")


args = get_args()

np.random.seed(args.seed)
torch.manual_seed(args.seed)

if args.cuda:
    torch.cuda.manual_seed(args.seed)


def get_local_map_boundaries(agent_loc, local_sizes,
full_sizes):
    loc_r, loc_c = agent_loc
    local_w, local_h = local_sizes
    full_w, full_h = full_sizes

    if args.global_downscaling > 1:
        gx1, gy1 = loc_r - local_w // 2, loc_c - local_h // 2
        gx2, gy2 = gx1 + local_w, gy1 + local_h
        if gx1 < 0:
            gx1, gx2 = 0, local_w
        if gx2 > full_w:
            gx1, gx2 = full_w - local_w, full_w

        if gy1 < 0:
            gy1, gy2 = 0, local_h
        if gy2 > full_h:
            gy1, gy2 = full_h - local_h, full_h
    else:
        gx1.gx2, gy1, gy2 = 0, full_w, 0, full_h

    return [gx1, gx2, gy1, gy2]


def main():
    # Setup Logging
    log_dir = "{}/models/{}/".format(args.dump_location,
args.exp_name)
```

```
    dump_dir = "{}/dump/{}/".format(args.dump_location,
args.exp_name)

    if not os.path.exists(log_dir):
        os.makedirs(log_dir)

    if not os.path.exists("{}/images/".format(dump_dir)):
        os.makedirs("{}/images/".format(dump_dir))

    logging.basicConfig(
        filename=log_dir + 'train.log',
        level=logging.INFO)
    print("Dumping at {}".format(log_dir))
    print(args)
    logging.info(args)

    # Logging and loss variables
    num_scenes = args.num_processes
    num_episodes = int(args.num_episodes)
    device = args.device = torch.device("cuda:0" if args.cuda
else "cpu")
    policy_loss = 0

    best_cost = 100000
    costs = deque(maxlen=1000)
    exp_costs = deque(maxlen=1000)
    pose_costs = deque(maxlen=1000)

    g_masks = torch.ones(num_scenes).float().to(device)
    l_masks = torch.zeros(num_scenes).float().to(device)

    best_local_loss = np.inf
    best_g_reward = -np.inf

    if args.eval:
        traj_lengths = args.max_episode_length //
args.num_local_steps
        explored_area_log = np.zeros((num_scenes,
num_episodes, traj_lengths))
        explored_ratio_log = np.zeros((num_scenes,
num_episodes, traj_lengths))

    g_episode_rewards = deque(maxlen=1000)

    l_action_losses = deque(maxlen=1000)

    g_value_losses = deque(maxlen=1000)
    g_action_losses = deque(maxlen=1000)
    g_dist_entropies = deque(maxlen=1000)

    per_step_g_rewards = deque(maxlen=1000)

    g_process_rewards = np.zeros((num_scenes))
```

```
    # Starting environments
    torch.set_num_threads(1)
    envs = make_vec_envs(args)
    obs, infos = envs.reset()

    # Initialize map variables
    ### Full map consists of 4 channels containing the
following:
    ### 1. Obstacle Map
    ### 2. Exploread Area
    ### 3. Current Agent Location
    ### 4. Past Agent Locations

    torch.set_grad_enabled(False)

    # Calculating full and local map sizes
    map_size = args.map_size_cm // args.map_resolution
    full_w, full_h = map_size, map_size
    local_w, local_h = int(full_w / args.global_downscaling),
\
                       int(full_h / args.global_downscaling)

    # Initializing full and local map
    full_map = torch.zeros(num_scenes, 4, full_w,
full_h).float().to(device)
    local_map = torch.zeros(num_scenes, 4, local_w,
local_h).float().to(device)

    # Initial full and local pose
    full_pose = torch.zeros(num_scenes, 3).float().to(device)
    local_pose = torch.zeros(num_scenes, 3).float().to(device)

    # Origin of local map
    origins = np.zeros((num_scenes, 3))

    # Local Map Boundaries
    lmb = np.zeros((num_scenes, 4)).astype(int)

    ### Planner pose inputs has 7 dimensions
    ### 1-3 store continuous global agent location
    ### 4-7 store local map boundaries
    planner_pose_inputs = np.zeros((num_scenes, 7))

    def init_map_and_pose():
        full_map.fill_(0.)
        full_pose.fill_(0.)
        full_pose[:, :2] = args.map_size_cm / 100.0 / 2.0

        locs = full_pose.cpu().numpy()
        planner_pose_inputs[:, :3] = locs
        for e in range(num_scenes):
            r, c = locs[e, 1], locs[e, 0]
            loc_r, loc_c = [int(r * 100.0 /
args.map_resolution),
```

```
                            int(c * 100.0 /
args.map_resolution)]

            full_map[e, 2:, loc_r - 1:loc_r + 2, loc_c -
1:loc_c + 2] = 1.0

            lmb[e] = get_local_map_boundaries((loc_r, loc_c),
                                              (local_w,
local_h),
                                              (full_w,
full_h))

            planner_pose_inputs[e, 3:] = lmb[e]
            origins[e] = [lmb[e][2] * args.map_resolution /
100.0,
                          lmb[e][0] * args.map_resolution /
100.0, 0.]

        for e in range(num_scenes):
            local_map[e] = full_map[e, :, lmb[e, 0]:lmb[e, 1],
lmb[e, 2]:lmb[e, 3]]
            local_pose[e] = full_pose[e] - \

torch.from_numpy(origins[e]).to(device).float()

    init_map_and_pose()

    # Global policy observation space
    g_observation_space = gym.spaces.Box(0, 1,
                                         (8,
                                          local_w,
                                          local_h),
dtype='uint8')

    # Global policy action space
    g_action_space = gym.spaces.Box(low=0.0, high=1.0,
                                    shape=(2,),
dtype=np.float32)

    # Local policy observation space
    l_observation_space = gym.spaces.Box(0, 255,
                                         (3,
                                          args.frame_width,
                                          args.frame_width),
dtype='uint8')

    # Local and Global policy recurrent layer sizes
    l_hidden_size = args.local_hidden_size
    g_hidden_size = args.global_hidden_size

    # slam
    nslam_module = Neural_SLAM_Module(args).to(device)
    slam_optimizer = get_optimizer(nslam_module.parameters(),
                                   args.slam_optimizer)
```

```
    # Global policy
    g_policy = RL_Policy(g_observation_space.shape,
g_action_space,
                         base_kwargs={'recurrent':
args.use_recurrent_global,
                                      'hidden_size':
g_hidden_size,
                                      'downscaling':
args.global_downscaling
                                      }).to(device)
    g_agent = algo.PPO(g_policy, args.clip_param,
args.ppo_epoch,
                       args.num_mini_batch,
args.value_loss_coef,
                       args.entropy_coef, lr=args.global_lr,
eps=args.eps,
                       max_grad_norm=args.max_grad_norm)

    # Local policy
    l_policy = Local_IL_Policy(l_observation_space.shape,
envs.action_space.n,

recurrent=args.use_recurrent_local,
                               hidden_size=l_hidden_size,

deterministic=args.use_deterministic_local).to(device)
    local_optimizer = get_optimizer(l_policy.parameters(),
                                    args.local_optimizer)

    # Storage
    g_rollouts = GlobalRolloutStorage(args.num_global_steps,
                                      num_scenes,
g_observation_space.shape,
                                      g_action_space,
g_policy.rec_state_size,
                                      1).to(device)

    slam_memory = FIFOMemory(args.slam_memory_size)

    # Loading model
    if args.load_slam != "0":
        print("Loading slam {}".format(args.load_slam))
        state_dict = torch.load(args.load_slam,
                                map_location=lambda storage,
loc: storage)
        nslam_module.load_state_dict(state_dict)

    if not args.train_slam:
        nslam_module.eval()

    if args.load_global != "0":
        print("Loading global {}".format(args.load_global))
        state_dict = torch.load(args.load_global,
```

```
                                map_location=lambda storage,
loc: storage)
        g_policy.load_state_dict(state_dict)

    if not args.train_global:
        g_policy.eval()

    if args.load_local != "0":
        print("Loading local {}".format(args.load_local))
        state_dict = torch.load(args.load_local,
                                map_location=lambda storage,
loc: storage)
        l_policy.load_state_dict(state_dict)

    if not args.train_local:
        l_policy.eval()

    # Predict map from frame 1:
    poses = torch.from_numpy(np.asarray(
        [infos[env_idx]['sensor_pose'] for env_idx
         in range(num_scenes)])
    ).float().to(device)

    _, _, local_map[:, 0, :, :], local_map[:, 1, :, :], _,
local_pose = \
        nslam_module(obs, obs, poses, local_map[:, 0, :, :],
                     local_map[:, 1, :, :], local_pose)

    # Compute Global policy input
    locs = local_pose.cpu().numpy()
    global_input = torch.zeros(num_scenes, 8, local_w,
local_h)
    global_orientation = torch.zeros(num_scenes, 1).long()

    for e in range(num_scenes):
        r, c = locs[e, 1], locs[e, 0]
        loc_r, loc_c = [int(r * 100.0 / args.map_resolution),
                        int(c * 100.0 / args.map_resolution)]

        local_map[e, 2:, loc_r - 1:loc_r + 2, loc_c - 1:loc_c
+ 2] = 1.
        global_orientation[e] = int((locs[e, 2] + 180.0) / 5.)

    global_input[:, 0:4, :, :] = local_map.detach()
    global_input[:, 4:, :, :] =
nn.MaxPool2d(args.global_downscaling)(full_map)

    g_rollouts.obs[0].copy_(global_input)
    g_rollouts.extras[0].copy_(global_orientation)

    # Run Global Policy (global_goals = Long-Term Goal)
    g_value, g_action, g_action_log_prob, g_rec_states = \
        g_policy.act(
            g_rollouts.obs[0],
```

```
            g_rollouts.rec_states[0],
            g_rollouts.masks[0],
            extras=g_rollouts.extras[0],
            deterministic=False
        )

    cpu_actions = nn.Sigmoid()(g_action).cpu().numpy()
    global_goals = [[int(action[0] * local_w), int(action[1] *
local_h)]
                    for action in cpu_actions]

    # Compute planner inputs
    planner_inputs = [{} for e in range(num_scenes)]
    for e, p_input in enumerate(planner_inputs):
        p_input['goal'] = global_goals[e]
        p_input['map_pred'] = global_input[e, 0, :,
:].detach().cpu().numpy()
        p_input['exp_pred'] = global_input[e, 1, :,
:].detach().cpu().numpy()
        p_input['pose_pred'] = planner_pose_inputs[e]

    # Output stores local goals as well as the the ground-
truth action
    output = envs.get_short_term_goal(planner_inputs)

    last_obs = obs.detach()
    local_rec_states = torch.zeros(num_scenes,
l_hidden_size).to(device)
    start = time.time()

    total_num_steps = -1
    g_reward = 0

    torch.set_grad_enabled(False)

    for ep_num in range(num_episodes):
        for step in range(args.max_episode_length):
            total_num_steps += 1

            g_step = (step // args.num_local_steps) %
args.num_global_steps
            eval_g_step = step // args.num_local_steps + 1
            l_step = step % args.num_local_steps

            # ------------------------------------------------------------
------------------------
            # Local Policy
            del last_obs
            last_obs = obs.detach()
            local_masks = l_masks
            local_goals = output[:, :-1].to(device).long()

            if args.train_local:
                torch.set_grad_enabled(True)
```

```
            action, action_prob, local_rec_states = l_policy(
                obs,
                local_rec_states,
                local_masks,
                extras=local_goals,
            )

            if args.train_local:
                action_target = output[:, -
1].long().to(device)
                policy_loss +=
nn.CrossEntropyLoss()(action_prob, action_target)
                torch.set_grad_enabled(False)
            l_action = action.cpu()
            # ------------------------------------------------------------------
------------------------

            # ------------------------------------------------------------------
------------------------
            # Env step
            obs, rew, done, infos = envs.step(l_action)

            l_masks = torch.FloatTensor([0 if x else 1
                                         for x in
done]).to(device)
            g_masks *= l_masks
            # ------------------------------------------------------------------
------------------------

            # ------------------------------------------------------------------
------------------------
            # Reinitialize variables when episode ends
            if step == args.max_episode_length - 1:  # Last
episode step
                init_map_and_pose()
                del last_obs
                last_obs = obs.detach()
            # ------------------------------------------------------------------
------------------------

            # ------------------------------------------------------------------
------------------------
            # Neural SLAM Module
            if args.train_slam:
                # Add frames to memory
                for env_idx in range(num_scenes):
                    env_obs = obs[env_idx].to("cpu")
                    env_poses = torch.from_numpy(np.asarray(
                        infos[env_idx]['sensor_pose']
                    )).float().to("cpu")
                    env_gt_fp_projs =
torch.from_numpy(np.asarray(
                        infos[env_idx]['fp_proj']
```

```
                    )).unsqueeze(0).float().to("cpu")
                    env_gt_fp_explored =
torch.from_numpy(np.asarray(
                        infos[env_idx]['fp_explored']
                    )).unsqueeze(0).float().to("cpu")
                    env_gt_pose_err =
torch.from_numpy(np.asarray(
                        infos[env_idx]['pose_err']
                    )).float().to("cpu")
                    slam_memory.push(
                        (last_obs[env_idx].cpu(), env_obs,
env_poses),
                        (env_gt_fp_projs, env_gt_fp_explored,
env_gt_pose_err))

            poses = torch.from_numpy(np.asarray(
                [infos[env_idx]['sensor_pose'] for env_idx
                 in range(num_scenes)])
            ).float().to(device)

            _, _, local_map[:, 0, :, :], local_map[:, 1, :,
:], _, local_pose = \
                nslam_module(last_obs, obs, poses,
local_map[:, 0, :, :],
                             local_map[:, 1, :, :],
local_pose, build_maps=True)

            locs = local_pose.cpu().numpy()
            planner_pose_inputs[:, :3] = locs + origins
            local_map[:, 2, :, :].fill_(0.)  # Resetting
current location channel
            for e in range(num_scenes):
                r, c = locs[e, 1], locs[e, 0]
                loc_r, loc_c = [int(r * 100.0 /
args.map_resolution),
                                int(c * 100.0 /
args.map_resolution)]

                local_map[e, 2:, loc_r - 2:loc_r + 3, loc_c -
2:loc_c + 3] = 1.
            # ------------------------------------------------------------------
------------------------

            # ------------------------------------------------------------------
------------------------
            # Global Policy
            if l_step == args.num_local_steps - 1:
                # For every global step, update the full and
local maps
                for e in range(num_scenes):
                    full_map[e, :, lmb[e, 0]:lmb[e, 1], lmb[e,
2]:lmb[e, 3]] = \
                        local_map[e]
                    full_pose[e] = local_pose[e] + \
```

```
torch.from_numpy(origins[e]).to(device).float()

                    locs = full_pose[e].cpu().numpy()
                    r, c = locs[1], locs[0]
                    loc_r, loc_c = [int(r * 100.0 /
args.map_resolution),
                                    int(c * 100.0 /
args.map_resolution)]

                    lmb[e] = get_local_map_boundaries((loc_r,
loc_c),

(local_w, local_h),
                                                      (full_w,
full_h))

                    planner_pose_inputs[e, 3:] = lmb[e]
                    origins[e] = [lmb[e][2] *
args.map_resolution / 100.0,
                                  lmb[e][0] *
args.map_resolution / 100.0, 0.]

                    local_map[e] = full_map[e, :,
                                   lmb[e, 0]:lmb[e, 1], lmb[e,
2]:lmb[e, 3]]
                    local_pose[e] = full_pose[e] - \

torch.from_numpy(origins[e]).to(device).float()

                locs = local_pose.cpu().numpy()
                for e in range(num_scenes):
                    global_orientation[e] = int((locs[e, 2] +
180.0) / 5.)
                global_input[:, 0:4, :, :] = local_map
                global_input[:, 4:, :, :] = \

nn.MaxPool2d(args.global_downscaling)(full_map)

                if False:
                    for i in range(4):
                        ax[i].clear()
                        ax[i].set_yticks([])
                        ax[i].set_xticks([])
                        ax[i].set_yticklabels([])
                        ax[i].set_xticklabels([])

ax[i].imshow(global_input.cpu().numpy()[0, 4 + i])
                    plt.gcf().canvas.flush_events()
                    # plt.pause(0.1)
                    fig.canvas.start_event_loop(0.001)
                    plt.gcf().canvas.flush_events()

                # Get exploration reward and metrics
```

```
                g_reward = torch.from_numpy(np.asarray(
                    [infos[env_idx]['exp_reward'] for env_idx
                     in range(num_scenes)])
                ).float().to(device)

                if args.eval:
                    g_reward = g_reward*50.0 # Convert reward
to area in m2

                g_process_rewards += g_reward.cpu().numpy()
                g_total_rewards = g_process_rewards * \
                                  (1 - g_masks.cpu().numpy())
                g_process_rewards *= g_masks.cpu().numpy()

per_step_g_rewards.append(np.mean(g_reward.cpu().numpy()))

                if np.sum(g_total_rewards) != 0:
                    for tr in g_total_rewards:
                        g_episode_rewards.append(tr) if tr !=
0 else None

                if args.eval:
                    exp_ratio = torch.from_numpy(np.asarray(
                        [infos[env_idx]['exp_ratio'] for
env_idx
                         in range(num_scenes)])
                    ).float()

                    for e in range(num_scenes):
                        explored_area_log[e, ep_num,
eval_g_step - 1] = \
                            explored_area_log[e, ep_num,
eval_g_step - 2] + \
                            g_reward[e].cpu().numpy()
                        explored_ratio_log[e, ep_num,
eval_g_step - 1] = \
                            explored_ratio_log[e, ep_num,
eval_g_step - 2] + \
                            exp_ratio[e].cpu().numpy()

                # Add samples to global policy storage
                g_rollouts.insert(
                    global_input,  g_rec_states,
                    g_action, g_action_log_prob, g_value,
                    g_reward, g_masks, global_orientation
                )

                # Sample long-term goal from global policy
                g_value, g_action, g_action_log_prob,
g_rec_states = \
                    g_policy.act(
                        g_rollouts.obs[g_step + 1],
                        g_rollouts.rec_states[g_step + 1],
                        g_rollouts.masks[g_step + 1],
```

```
                        extras=g_rollouts.extras[g_step + 1],
                        deterministic=False
                    )
                cpu_actions =
nn.Sigmoid()(g_action).cpu().numpy()
                global_goals = [[int(action[0] * local_w),
                                 int(action[1] * local_h)]
                                for action in cpu_actions]

                g_reward = 0
                g_masks =
torch.ones(num_scenes).float().to(device)
            # ------------------------------------------------------------------
------------------------

            # ------------------------------------------------------------------
------------------------
            # Get short term goal
            planner_inputs = [{} for e in range(num_scenes)]
            for e, p_input in enumerate(planner_inputs):
                p_input['map_pred'] = local_map[e, 0, :,
:].cpu().numpy()
                p_input['exp_pred'] = local_map[e, 1, :,
:].cpu().numpy()
                p_input['pose_pred'] = planner_pose_inputs[e]
                p_input['goal'] = global_goals[e]

            output = envs.get_short_term_goal(planner_inputs)
            # ------------------------------------------------------------------
------------------------

            ### TRAINING
            torch.set_grad_enabled(True)
            # ------------------------------------------------------------------
------------------------
            # Train Neural SLAM Module
            if args.train_slam and len(slam_memory) >
args.slam_batch_size:
                for _ in range(args.slam_iterations):
                    inputs, outputs =
slam_memory.sample(args.slam_batch_size)
                    b_obs_last, b_obs, b_poses = inputs
                    gt_fp_projs, gt_fp_explored, gt_pose_err =
outputs

                    b_obs = b_obs.to(device)
                    b_obs_last = b_obs_last.to(device)
                    b_poses = b_poses.to(device)

                    gt_fp_projs = gt_fp_projs.to(device)
                    gt_fp_explored = gt_fp_explored.to(device)
                    gt_pose_err = gt_pose_err.to(device)
```

```
                    b_proj_pred, b_fp_exp_pred, _, _,
b_pose_err_pred, _ = \
                        nslam_module(b_obs_last, b_obs,
b_poses,
                                     None, None, None,
                                     build_maps=False)
                    loss = 0
                    if args.proj_loss_coeff > 0:
                        proj_loss =
F.binary_cross_entropy(b_proj_pred,

gt_fp_projs)
                        costs.append(proj_loss.item())
                        loss += args.proj_loss_coeff *
proj_loss

                    if args.exp_loss_coeff > 0:
                        exp_loss =
F.binary_cross_entropy(b_fp_exp_pred,

gt_fp_explored)
                        exp_costs.append(exp_loss.item())
                        loss += args.exp_loss_coeff * exp_loss

                    if args.pose_loss_coeff > 0:
                        pose_loss =
torch.nn.MSELoss()(b_pose_err_pred,

gt_pose_err)
                        pose_costs.append(args.pose_loss_coeff
*
                                          pose_loss.item())
                        loss += args.pose_loss_coeff *
pose_loss

                    if args.train_slam:
                        slam_optimizer.zero_grad()
                        loss.backward()
                        slam_optimizer.step()

                    del b_obs_last, b_obs, b_poses
                    del gt_fp_projs, gt_fp_explored,
gt_pose_err
                    del b_proj_pred, b_fp_exp_pred,
b_pose_err_pred

            # ------------------------------------------------------------------
------------------------

            # ------------------------------------------------------------------
------------------------
            # Train Local Policy
            if (l_step + 1) % args.local_policy_update_freq ==
0 \
```

```
                    and args.train_local:
                local_optimizer.zero_grad()
                policy_loss.backward()
                local_optimizer.step()
                l_action_losses.append(policy_loss.item())
                policy_loss = 0
                local_rec_states = local_rec_states.detach_()
            # ------------------------------------------------------------
------------------------

            # ------------------------------------------------------------
------------------------
            # Train Global Policy
            if g_step % args.num_global_steps ==
args.num_global_steps - 1 \
                    and l_step == args.num_local_steps - 1:
                if args.train_global:
                    g_next_value = g_policy.get_value(
                        g_rollouts.obs[-1],
                        g_rollouts.rec_states[-1],
                        g_rollouts.masks[-1],
                        extras=g_rollouts.extras[-1]
                    ).detach()

                    g_rollouts.compute_returns(g_next_value,
args.use_gae,
                                               args.gamma,
args.tau)
                    g_value_loss, g_action_loss,
g_dist_entropy = \
                        g_agent.update(g_rollouts)
                    g_value_losses.append(g_value_loss)
                    g_action_losses.append(g_action_loss)
                    g_dist_entropies.append(g_dist_entropy)
                g_rollouts.after_update()
            # ------------------------------------------------------------
------------------------

            # Finish Training
            torch.set_grad_enabled(False)
            # ------------------------------------------------------------
------------------------

            # ------------------------------------------------------------
------------------------
            # Logging
            if total_num_steps % args.log_interval == 0:
                end = time.time()
                time_elapsed = time.gmtime(end - start)
                log = " ".join([
                    "Time:
{0:0=2d}d".format(time_elapsed.tm_mday - 1),
                    "{},".format(time.strftime("%Hh %Mm %Ss",
time_elapsed)),
```

```
                    "num timesteps {},".format(total_num_steps
*
                                                num_scenes),
                    "FPS {},".format(int(total_num_steps *
num_scenes \
                                         / (end - start)))
                ])

                log += "\n\tRewards:"

                if len(g_episode_rewards) > 0:
                    log += " ".join([
                        " Global step mean/med rew:",
                        "{:.4f}/{:.4f},".format(
                            np.mean(per_step_g_rewards),
                            np.median(per_step_g_rewards)),
                        " Global eps mean/med/min/max eps
rew:",
                        "{:.3f}/{:.3f}/{:.3f}/{:.3f},".format(
                            np.mean(g_episode_rewards),
                            np.median(g_episode_rewards),
                            np.min(g_episode_rewards),
                            np.max(g_episode_rewards))
                    ])

                log += "\n\tLosses:"

                if args.train_local and len(l_action_losses) >
0:
                    log += " ".join([
                        " Local Loss:",
                        "{:.3f},".format(
                            np.mean(l_action_losses))
                    ])

                if args.train_global and len(g_value_losses) >
0:
                    log += " ".join([
                        " Global Loss value/action/dist:",
                        "{:.3f}/{:.3f}/{:.3f},".format(
                            np.mean(g_value_losses),
                            np.mean(g_action_losses),
                            np.mean(g_dist_entropies))
                    ])

                if args.train_slam and len(costs) > 0:
                    log += " ".join([
                        " SLAM Loss proj/exp/pose:"
                        "{:.4f}/{:.4f}/{:.4f}".format(
                            np.mean(costs),
                            np.mean(exp_costs),
                            np.mean(pose_costs))
                    ])
```

```
                print(log)
                logging.info(log)
            # ------------------------------------------------------------------
------------------------

            # ------------------------------------------------------------------
------------------------
            # Save best models
            if (total_num_steps * num_scenes) %
args.save_interval < \
                    num_scenes:

                # Save Neural SLAM Model
                if len(costs) >= 1000 and np.mean(costs) <
best_cost \
                        and not args.eval:
                    best_cost = np.mean(costs)
                    torch.save(nslam_module.state_dict(),
                               os.path.join(log_dir,
"model_best.slam"))

                # Save Local Policy Model
                if len(l_action_losses) >= 100 and \
                        (np.mean(l_action_losses) <=
best_local_loss) \
                        and not args.eval:
                    torch.save(l_policy.state_dict(),
                               os.path.join(log_dir,
"model_best.local"))

                    best_local_loss = np.mean(l_action_losses)

                # Save Global Policy Model
                if len(g_episode_rewards) >= 100 and \
                        (np.mean(g_episode_rewards) >=
best_g_reward) \
                        and not args.eval:
                    torch.save(g_policy.state_dict(),
                               os.path.join(log_dir,
"model_best.global"))
                    best_g_reward = np.mean(g_episode_rewards)

            # Save periodic models
            if (total_num_steps * num_scenes) %
args.save_periodic < \
                    num_scenes:
                step = total_num_steps * num_scenes
                if args.train_slam:
                    torch.save(nslam_module.state_dict(),
                               os.path.join(dump_dir,

"periodic_{}.slam".format(step)))
                if args.train_local:
                    torch.save(l_policy.state_dict(),
```

```
                                os.path.join(dump_dir,

"periodic_{}.local".format(step)))
                if args.train_global:
                    torch.save(g_policy.state_dict(),
                               os.path.join(dump_dir,

"periodic_{}.global".format(step)))
            # ------------------------------------------------------------------
------------------------

    # Print and save model performance numbers during
evaluation
    if args.eval:
        logfile =
open("{}/explored_area.txt".format(dump_dir), "w+")
        for e in range(num_scenes):
            for i in range(explored_area_log[e].shape[0]):
                logfile.write(str(explored_area_log[e, i]) +
"\n")
                logfile.flush()

        logfile.close()

        logfile =
open("{}/explored_ratio.txt".format(dump_dir), "w+")
        for e in range(num_scenes):
            for i in range(explored_ratio_log[e].shape[0]):
                logfile.write(str(explored_ratio_log[e, i]) +
"\n")
                logfile.flush()

        logfile.close()

        log = "Final Exp Area: \n"
        for i in range(explored_area_log.shape[2]):
            log += "{:.5f}, ".format(
                np.mean(explored_area_log[:, :, i]))

        log += "\nFinal Exp Ratio: \n"
        for i in range(explored_ratio_log.shape[2]):
            log += "{:.5f}, ".format(
                np.mean(explored_ratio_log[:, :, i]))

        print(log)
        logging.info(log)


if_name_== "_main__":
    main()
```

5. ВЫВОДЫ

В рамках домашнего задания был выполнен обзор теоретических и практических материалов, связанных со статьей «learning to explore using Active neural slam».

В статье рассматривался SLAM алгоритм, который был улучшен для более эффективного использования в задачах построения маршрута от точки до точки.

В практической части был выполнен обзор содержимого репозитория авторов статьи на GitHub.

6. СПИСОК ИСТОЧНИКОВ

1. Browse State-of-the-Art. – Текст. Изображение: электронные // Papers With Code : [сайт]. – URL: https://paperswithcode.com/sota (дата обращения: 05.06.2022).
2. Learning to Explore using Active Neural SLAM. – Текст. Изображение: электронные // Papers With Code : [сайт]. – URL: https://paperswithcode.com/paper/learning-to-explore-using-active-neural-slam#code (дата обращения: 05.06.2022).
3. Веб-сайт проекта «Learning to Explore using Active Neural SLAM.» [сайт]. – URL: https://devendrachaplot.github.io/projects/Neural-SLAM (дата обращения: 05.06.2022).
4. https://github.com/devendrachaplot/Neural-SLAM