ISSN 0868–5886 *НАУЧНОЕ ПРИБОРОСТРОЕНИЕ, 2023, том 33, № 1, с. 95–108*

МАТЕМАТИЧЕСКИЕ МЕТОДЫ И МОДЕЛИРОВАНИЕ В ПРИБОРОСТРОЕНИИ

УДК 004.05, 004.932.2

© А. Г. Лапушкин, Д. А. Гаврилов, О. А. Поткин, 2023

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ СОЗДАНИЯ СИНТЕЗИРОВАННЫХ ДАННЫХ И СИМУЛЯТОР С ОБРАТНОЙ СВЯЗЬЮ ДЛЯ ТЕСТИРОВАНИЯ АЛГОРИТМОВ МАШИННОГО ОБУЧЕНИЯ

В настоящей работе представлено программное обеспечение, позволяющее создавать новые данные для обучения и проверять уже имеющиеся алгоритмы в режиме симуляции. Разработанная программа позволяет производить наборы связанных данных, в том числе совместные наборы данных видимого и инфракрасного диапазонов, полученных с помощью одной камеры или стереопары, дополнительной информации в виде лидарных данных или карты глубины, сегментационной картины, данных о расположении интересующих объектов на фото- или видеоизображении. Структура разработанного программного обеспечения позволяет осуществлять дальнейшее усовершенствование подходов и возможностей доработки получившегося конвейера под разные цели и задачи.

Кл. сл.: машинное обучение, обучающие выборки, обучение нейросетевых алгоритмов, тестирование алгоритмов, симулятор

ВВЕДЕНИЕ

Разработчики алгоритмов обучения нейросетей рано или поздно сталкиваются с кризисом, вызванным отсутствием достаточного количества размеченных визуальных данных [1]. Как правило, ни одно исследование в машинном обучении не обходится без экспериментов на реальных или модельных данных, подтверждающих практическую работоспособность метода. Существуют различные методы решения этой проблемы, однако универсальные методы на данный момент не обнаружены [2–4]. Кроме того, при разработке алгоритмов важное значение имеет возможность осуществления их тестирования [5]. В настоящей статье речь пойдет о специализированном программном комплексе, который позволяет одновременно создавать новые данные для обучения и проверять уже имеющиеся алгоритмы в режиме симуляции. Необходимость разработки такого программного обеспечения обусловлена требованием получения связанных данных для решения некоторых задач. Речь идет о совместных наборах данных видимого и инфракрасного диапазонов, полученных с помощью одной камеры или стереопары, и дополнительной информации в виде лидарных данных или просто карты глубины, сегментационной картины, данных о расположении интересующих объектов на фото- или видеоизображении. Детальный анализ различных открытых источников показал, что подобных наборов данных на текущий момент не существует [6]. В результате работы создано программное обеспечение, позволяющее синтезировать данные для генерации обучающих выборок. Разработанные описания внутренней структуры данного программного обеспечения позволяют осуществлять дальнейшее усовершенствование подходов и возможностей доработки получившегося конвейера под разные цели и задачи.

ОПИСАНИЕ РАБОТЫ СИМУЛЯТОРА

Разработанный симулятор предназначен для решения проблем, связанных с обучением и тестированием движения беспилотного роботизированного транспортного средства. Основные функции симулятора:

– проведение *software-in-the-loop* и *hardware-in-the-loop* тестирований алгоритмов, использующихся в аппаратуре роботехнических комплексов (РТК) в режиме "замедленного реального времени";

– формирование изображений сцены в видимом и инфракрасном диапазонах вместе с эталонной разметкой для разработки и обучения нейросетевых алгоритмов распознавания объектов и семантической сегментации сцены;

– отладка алгоритмов стереореконструкции, визуальной одометрии и SLAM.

Симулятор имитирует такие источники выходных данных сенсоров, включая эффекты их несовершенства, как: телевизионные (ТВ) и тепловизорные (ТПВ) камеры, лидары, спутниковые навигационные системы, инерциальные системы (ИНС), колесная одометрия наземного колесного робототехнического транспортного средства (РТК), датчик угла поворота колес РТК.

Представленный программный компонент поддерживает несколько сценариев работы.

Режим генератора позволяет сгенерировать наборы связанных данных для обучения алгоритмов. Данные генерируются параллельно на нескольких GPU кластера и сохраняются в памяти кластера.

Тестовый режим позволяет тестировать отдельные фрагменты набора данных в режиме реального времени с помощью веб-интерфейса.

Режим полигона представляет собой тестовый полигон, по которому может двигаться некоторая физическая модель, например беспилотный летательный аппарат или автомобиль. В этом режиме цикл замыкается обратной связью, позволяя протестировать алгоритмы в условиях, близких к реальным, без последствий для физической реализации.

Ручной режим позволяет перехватывать управление физической моделью оператором с целью корректировки во время испытаний. Программа симулирует последовательный видеоряд и передает с помощью сетевых сокетов данные дальше на демонстрационный компьютер с оборудованием, а также в другой кластер, который занимается распознаванием симулированных изображений и обеспечивает обратную связь. При этом во время тестирования беспилотного транспортного средства нет необходимости привлекать человека-оператора для дополнительного контроля за происходящим, так что отсутствует опасность физического воздействия [7, 8], более того, для быстрой проверки гипотез возможен запуск нескольких сценариев одновременно.

Схема симулятора представлена на рис. 1.

Разработанное программное обеспечение представляет собой универсальную реализацию симулятора-генератора. Внутренне — это набор моделей, шейдерных программ и скриптов, связанных между собой в единое представление. Режимы программного комплекса и внутреннее состояние сцен выставляются с помощью конфигурационного *json*-файла.

Рис. 1. Схема симулятора

В тестовом режиме в конфигурации сохраняются две локальные папки. В первой находятся описания кадров в формате *json*, во вторую поступают сгенерированные по этим описаниям файлы. Благодаря тестовому режиму становится возможным предпросмотр крупных сцен без траты времени на создание обширной сборки.

Режим генератора позволяет получить обширную случайную выборку с помощью рандомизации положения камеры и объектов. Описание сцены также имеет формат *json*. Описание попадает в программное обеспечение через специализированную очередь. Происходит отрисовка визуальной картинки, маски расположения объектов интереса и отдельно описание кадра с наименованиями объектов и их позиций в пиксельных координатах. Генерирование большого количества данных может занимать достаточно долгое время, однако возможно распараллеливание задач. На кластере имеются 8 GPU, каждый из которых задействован в генерировании большой выборки. Для управления генератором используется веб-приложение. В нем можно сформировать задание, указать количество GPU, которые должны быть задействованы. Статус генерирования выборки, каждый шаг генерации отправляется в веб-приложение. После завершения процесса генерации с помощью веб-приложения возможно получить сгенерированные данные по прямой ссылке. Схема генератора представлена на рис. 2.

ВИРТУАЛЬНАЯ ОПТИКОЭЛЕКТРОННАЯ СИСТЕМА ДЛЯ ЭКСПЕРИМЕНТОВ С АЛГОРИТМАМИ

Оптикоэлектронная система (ОЭС), используемая для проведения экспериментов с алгоритмами, представляет собой идеально жесткую сборку из 4 стереопар. Оптические оси камер лежат в одной плоскости. Углы азимута оптических осей стереопар составляют 0, 90, 180, 270°. Горизонтальное поле зрения каждой стереопары — 94°, что обеспечивает пересечение полей зрения. Здесь следует заметить, что данное значение является типичным. В целях конкретных экспериментов данное значение может быть изменено (например, с целью оптимизации работы алгоритмов). Вертикальное поле зрение каждой стереопары — 70° (также является типичным и может варьироваться). Расстояние между оптическими осями камер, входящих в стереопару (база), — 1 м (может варьироваться). Оптикоэлектронная система позволяет изменять параметры сцены с помощью входных параметров. Расстояние между камерами и параметры камер, такие как фокусное расстояние, размер оптического сенсора, регулируются с помощью конфигурационного файла.

Формат выходных данных представляет собой восемь изображений с камер, координаты местоположения центра автомобиля, его поворот в глобальной

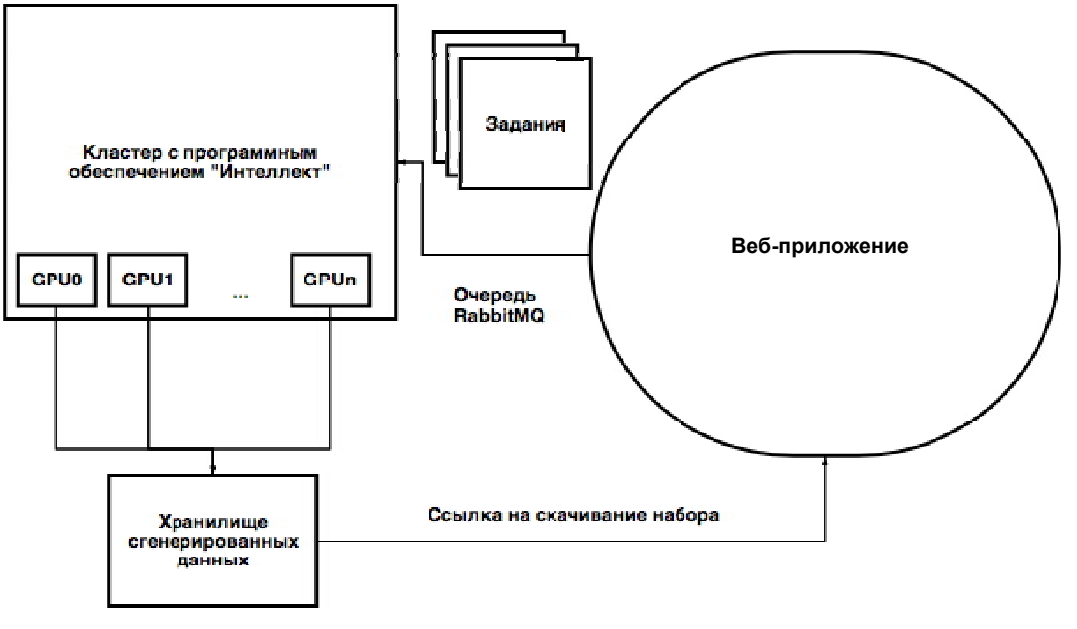

Рис. 2. Схема генератора

системе координат. Также известны положения и повороты камер в локальном пространстве автомобиля. Выходные данные представляются в виде структуры папок, в которые помещаются описания сцены в формате *json* и изображения с камер.

ОПИСАНИЕ ФОРМАТА КОНФИГУРАЦИОННЫХ ДАННЫХ ДЛЯ ГЕНЕРАТОРА

Структура данных для генератора

Генератор должен получить исчерпывающую информацию о типах объектов в кадре, расположении объектов, углах освещения, необходимости занесения объектов в описание сцены и т.д. Данные отправляются в программное обеспечение в формате *json* и имеют следующую структуру:

```
{
    "Count": 1,
    "Comment": "",
    "Map": "Test_Track_00001",
    "BackgroundObjects": [_],
    "Foreground Objects": [_]
    "Cameras": [_]
    "Images": [_]
    "Environment": {_}
    "NOPlacements": [_]
    "DOPlacements": [_]
    "Sensors": [_]
}.
```

Описание полей Count, Comment, Map, BackgroundObjects, Foreground Objects в данных генератора

– Поле **Count** хранит количество кадров, которые необходимо будет сделать генератору.

– Поле **Comment** может содержать поясняющего характера комментарий к кадру / набору кадров.

– Поле **Map** хранит строку, совпадающую по названию с одной из статических карт, заранее подготовленных и настроенных для использования.

– Поле **BackgroundObjects** хранит массив закадровых неподвижных объектов, для которых не планируется менять позицию, однако с помощью которых можно привнести в сцену некоторые отличия.

– Поле **Foreground Objects** это массив строк с именами так называемых объектов переднего плана. Такие объекты являются "действующими лицами" сцены, обычно на них установлены камеры. Это может быть автомобиль со всеми его датчиками, но кроме автомобиля это может быть снимающий автомобиль сверху квадрокоптер или другой иной беспилотник.

Описание поля Images в данных генератора

Поле **Images** содержит перечисление кадров, которые необходимо сделать всем камерам на сцене. Обычно это снимок в видимом спектре, инфракрасном и снимок сегментационной камеры, позволяющей запечатлеть объекты на сцене в виде однотонных пятен.

Примерный вид содержания поля **Images**:

```
"Images": [
    {
        "Tag": "image",
        "ImageType": "Visible",
        "Camera": 0
    },
    {
        "Tag": "mask",
        "ImageType": "Mask",
        "Camera": 0
    }
],
```

Описание поля Environment в данных генератора

Поле **Environment** содержит в себе структуру, определяющую параметры освещения на сцене. Структура описывает поведение направленного источника освещения. Это не значит, что сцена не может содержать других типов освещения, однако добавлять их нужно заранее в статическую составляющую сцены.

Структура сцены в поле **Environment:**

```
"Environment": {
    "LightIntensity": 1,
    "AmbientIntensity": 1,
    "ShadowPitch": 49,
    "ShadowRoll": -4,
    "ShadowIntensity": 1,
    "ShadowEnabled": true
},
```

Поля описания сцены определяют параметры освещения на сцене.

– Поле **LightIntensity** отвечает за интенсивность направленного источника освещения.

– Поле **AmbientIntensity** позволяет выбрать интенсивность рассеянного окружающего света.

– Углы направленного источника освещения задаются с помощью полей **ShadowPitch** и **ShadowRoll**.

– Поле **ShadowIntensity** меняется от 0.0 до 1.0 и определяет яркость тени.

На этот параметр также может влиять чрезмерный **AmbientIntensity**.

– Поле **ShadowEnabled** позволяет включать и отключать тени от направленного источника освещения.

Описание поля NOPlacements в данных генератора

Поле **NOPlacements** представляет собой массив структур описания движущихся объектов на сцене.

Содержание поля **NOPlacements**:

```
"NOPlacements": [
    {
        "Id": "fg0",
        "ObjectPlacement": {
            "PlacementType": "absolute",
            "ParentId": null,
            "Position": {
                "X": 14223.819710696813,
                "Y": -3410.0956920878875,
                "Z": 100,
                "Yaw": 90,
                "Pitch": 0,
                "Roll": 0
            },
            "Scale": {
                "ScaleX": 1,
                "ScaleY": 1,
                "ScaleZ": 1
            },
            "Model": {}
        }
    },
    -
],
```

– Структура имеет поле **Id**, служащее псевдонимом для указанного объекта.

– Поле **ObjectPlacement** содержит структуру, описывающую расстановку динамических объектов.

– Поле **PlacementType** описывает способ расстановки. Это может быть **absolute**, т.е. указывается позиция объекта в глобальной системе координат. Поле **PlacementType**, выставленное в **relative**, позволяет выставлять объекты относительно указанного объекта.

– Поле **ParentId** позволяет выставить для текущего объекта родительский объект, относительно которого будет отсчитываться позиция для режима **relative**.

– Поле **Position** представляет собой структуру, описывающую позицию расстановки объекта. В данной структуре заданы параметры позиции, указанные с помощью полей **X**, **Y**, **Z**, и также поворот объекта записывается в эйлеровых координатах с помощью полей **Yaw**, **Pitch**, **Roll**, указываемых в градусах.

– Поле **Scale** содержит структуру, которая описывает масштаб динамического объекта по трем осям с помощью параметров **ScaleX**, **ScaleY**, **ScaleZ**.

– Поле **Model** содержит структуру, которая позволяет явно указать, какую модель движения должен избрать объект в динамике.

Описание поля Cameras в данных генератора

Поле **Cameras** представляет собой массив, в котором содержатся объекты настройки каждой камеры на сцене по отдельности.

Внешний вид настройки камеры:

```
"Cameras": [
{
    "PixelSizeX": 3.45e-06,
    "PixelSizeY": 3.45e-06,
    "FocalLength": 0.01875,
    "MatrixW": 6144,
    "MatrixH": 6144,
    "CameraMainOffset": 0.0,
    "CameraCrossOffset": 0.0,
    "CameraAxisAngle": 0.0,
    "ImageFormat": "png",
    "ObjectId": "fg0",
    "CameraId": "forward_cam_0",
    "IsOrtho": false,
    "OrthoSize": 200
    "ImageEnhancementParameters": {
        "CalibrationRulerOn": false,
        "CalibrationRulerDistance": 0.0,
        "DownScaleSimulationOn": false,
        "FrameScaleX": 0.0,
        "FrameScaleY": 0.0,
        "CameraShakingOn": false,
        "ExPositionTimeSeconds": 0.0,
        "NonIdealEffectsOn": false
        },
    },
    -
},
```

– В полях **PixelSizeX** и **PixelSizeY** выставляют размер пикселя в виртуальной камере, в данном случае выставлен стандартный размер пикселя 3.45 мкм, соответствующий реальной камере, установленной на автомобиле.

– Поле **FocalLength** относится к фокусному расстоянию камеры, реальная величина соответствует 0.01875 м.

– Поля **MatrixW** и **MatrixH** устанавливают размер выходного изображения в пикселях.

– Поля **CameraMainOffset**, **CameraCrossOffset** и **CameraAxisAngle** позволяют задать смещения камеры в метрах относительно центра съемки, а также поворот.

– Поле **ImageFormat** позволяет получить желаемый формат снимка.

– Поле **ObjectId** позволяет закрепить камеру за назначенным объектом.

– Поле **CameraId** позволяет присвоить камере уникальный идентификатор, по указанному в этом поле идентификатору к камере можно обращаться в дальнейшем.

– Поле **IsOrtho** позволяет переключить камеру в ортонормированный режим, при включении этого поля также необходимо указать настройки ортонормированного вида, например **OrthoSize** — размер ортонормированной области.

– Поле **ImageEnhancementParameters** хранит структуру, позволяющую описать постпроцессинг кадра. В кадр можно добавить калибровочную сетку с помощью активации поля **CalibrationRulerOn** и, в случае включения SerDe, можно указать расстояние от камеры в поле **CalibrationRulerDistance**.

– Поле **DownScaleSimulationOn** включает режим рендеринга кадра высокого разрешения с последующим масштабированием до указанных в полях **MatrixW** и **MatrixH** размеров с использованием заранее оговоренного алгоритма сжатия.

– Сжатие также может быть задано вручную с помощью полей **FrameScaleX** и **FrameScaleY**.

– Поле **CameraShakingOn** включает режим смазывания изображения при тряске. В этом режиме рисуются сразу несколько кадров, которые суммируются по некоторому, заранее оговоренному алгоритму. Для этого режима также необходимо задать время экспозиции с помощью поля **ExPositionTimeSeconds**.

– Поле **NonIdealEffectsOn** включает режим постобработки, который добавляет в кадр дефекты, характерные той или иной камере, с помощью заранее выбранного алгоритма.

Описание поля Sensors в данных генератора

Поле **Sensors** служит для предоставления информации о дополнительных тестирующих устройствах. Описание, реализация и взаимодействие устройства варьируются в зависимости от реализации генератора. При включении в список **Sensors** тестирующего оборудования, если такое есть в наличии, генератор дописывает в сопроводительную информацию для сгенерированного набора данных дополнительные данные с сенсора. В реализации генератора для беспилотного автомобиля присутствуют сенсоры лидара, задних и передних радаров, датчик одометрии колес, гироскопический датчик и акселерометр.

РЕЖИМ СИМУЛЯЦИИ ДАННЫХ

Генератор используется для единовременного в рамках каждого исследования формирования набора данных, которое требует относительно продолжительного времени. В дальнейшем в рамках данного исследования запуск генератора может не потребоваться.

В качестве сцены для экспериментов с алгоритмами используется виртуальный город с прилегающей сельской местностью.

Рис. 3. Пример изображения городской среды, сгенерированного с помощью разработанного фотореалистичного симулятора

Рис. 4. Пример панорамного снимка

Сцена состоит из нескольких десятков зданий разной этажности, вместе образующих городской квартал, и прилегающей сельской местности с зелеными насаждениями. Присутствуют перекрестки, участки с круговым движением, пешеходные переходы, знаки дорожного движения, места для парковки, ремонтирующиеся участки дороги. В городе осуществляется движение машин и пешеходов. Пример изображения городской среды, сгенерированного с помощью разработанного фотореалистичного симулятора, представлен на рис. 3. В то же время симулятор должен быть рассчитан на частое использование, поэтому в режиме симулятора предъявляются повышенные требования ко времени формирования кадра. Сцены, предназначенные для симулятора, должны быть оптимизированы, а постобработка кадров сокращена до минимума. Оптимизация касается в том числе данных датчиков.

Для оценки точности работы алгоритмов расчет метрики может осуществляться по сегментационной маске, однако генерирование маски часто нецелесообразно для симуляции. Для оценочных метрик используются менее затратные по производительности способы, например генерирование разметки с помощью проекции ограничивающего прямоугольника (*bounding box*).

Кроме максимальной оптимизации сцены и моделей также необходимо оптимизировать протокол обмена. В разработанном программном обеспечении для сохранения большей пропускной способности сетевых каналов принято решение подвергать сжатию изображения, генерированные симулятором. Большая часть информации накапливается с помощью камер, расположенных на крыше автомобиля. Предполагается использование восьми камер, выставленных таким образом, чтобы камеры захватывали области видимости своих соседей. Такое построение выбрано для последующего укомплектования изображений в панораму. Кроме того, симулятор обладает функционалом, позволяющим получать готовый панорамный снимок для дальнейшей обработки нейросетевыми алгоритмами. Пример панорамного снимка представлен на рис. 4.

ЗАКЛЮЧЕНИЕ

Представлено описание программного обеспечения для создания синтезированных данных и симулятор с обратной связью для тестирования алгоритмов машинного обучения. Разработанный симулятор предназначен для решения проблем, связанных с обучением и тестированием движения беспилотного роботизированного транспортного средства, и поддерживает режимы работы генератора данных, тестовый режим, режим тестового полигона и ручной режим. Описана структура разработанного программного обеспечения.

Представленное программное обеспечение позволяет осуществлять тестирование беспилотного транспортного средства без привлечения человека-оператора, т.е. отсутствует опасность физического воздействия. С его помощью возможен запуск нескольких сценариев одновременно для ускорения проверки гипотез.

Работа выполнена в рамках договора № 70-2021-00138 от 1 ноября 2021 г. с АНО "Аналитический центр при Правительстве Российской Федерации" на реализацию программы исследовательского центра в сфере искусственного интеллекта, ИГК 00000D730321P5Q0002.

СПИСОК ЛИТЕРАТУРЫ

1. *Medvedev M., Kadhim A., Brosalin D.* Developmentof the Neural-Based Navigation System for a Ground-Based Mobile Robot // 7th International Conference on Mecha-

tronics and Robotics Engineering, ICMRE 2021. 2021. P. 35–40. DOI: 10.1109/ICMRE51691.2021.9384825
2. *Кафтанников И.Л., Парасич А.В.* Проблемы формирования обучающей выборки в задачах машинного обучения // Вестник ЮУрГУ. Серия "Компьютерные технологии, управление, радиоэлектроника". 2016. Т. 16, № 3. С. 15–24. DOI: 10.14529/ctcr160302
3. *Roh Y., Heo G., Whang S.E.* A survey on data collection for machine learning: a big data AI integration perspective // IEEE Transactions on Knowledge and Data Engineering. 2021. Vol. 33, is. 4. P. 1328–1347 DOI: 10.1109/TKDE.2019.2946162
4. *Гаврилов Д.А., Щелкунов Н.Н.* Программное обеспечение разметки крупноформатных аэрокосмических изображений и подготовки обучающих выборок // Научное приборостроение. 2020. Т. 30, № 2. С. 67–75. URL: http://iairas.ru/mag/2020/abst2.php#abst9
5. *Гаврилов Д.А.* Программно-аппаратный комплекс тестирования алгоритмов детектирования и локализации объектов в видеопоследовательностях // Научное приборостроение. 2019. Т. 29, № 1. С. 149–156. URL: http://iairas.ru/mag/2019/abst1.php#abst22
6. *Лапушкин А.Г., Гаврилов Д.А., Щелкунов Н.Н., Бакеев Р.Н.* Основные подходы к подготовке визуальных данных для обучения нейросетевых алгоритмов // Искусственный интеллект и принятие решений. 2021. № 4. С. 62–74. DOI: 10.14357/20718594210406
7. *Писарева О.М., Алексеев В.А., Медников Д.Н., Стариковский А.В.* Характеристика зон уязвимости и источников угроз информационной безопасности эксплуатации беспилотных автомобилей в интеллектуальной транспортной системе // Научно-технические ведомости СПбГПУ. Экономические науки. 2021. Т. 14, № 4. С. 20–36. DOI: 10.18721/JE.14402
8. *Li L., Huang W.-L., Liu Y., Zheng N.-N., Wang F.-Y.* Intelligence testing for autonomous vehicles: a new approach // IEEE Transactions on Intelligent Vehicles. 2016. Vol. 1, is. 2. P. 158–166. DOI: 10.1109/TIV.2016.2608003

Московский физико-технический институт (национальный исследовательский университет), г. Долгопрудный (*Лапушкин А.Г., Гаврилов Д.А.*)

ООО СберАвтомотив Технологии, Москва (*Поткин О.А.*)

Контакты: *Лапушкин Андрей Георгиевич*, lapushkin.ag@mipt.ru

Материал поступил в редакцию 31.10.2022

SYNTHESIZED DATA CREATION SOFTWARE AND FEEDBACK SIMULATOR FOR TESTING MACHINE LEARNING ALGORITHMS

A. G. Lapushkin[1], D. A. Gavrilov[1], O. A. Potkin[2]

[1]*Moscow Institute of Physics and Technology (National Research University), Dolgoprudny, Russia*
[2]*Sber Automotive Technologies, Moscow, Russia*

This paper presents software that allows the creation of new training data and tests existing algorithms in simulation mode. The developed program allows obtaining related data sets, including combined sets of the visible and infrared ranges using a single camera or stereo pair, additional information in the form of lidar data or a depth map, a segmentation pattern, and data on the location of objects of interest in a photo or video image. The structure of the developed software allows for further improvement of the approaches and possibilities for finalizing the resulting pipeline for different purposes and tasks.

Keywords: machine learning, training sets, training of neural network algorithms, algorithm testing, simulator

INTRODUCTION

Developers of neural network training algorithms sooner or later face a crisis caused by the lack of sufficient marked-up visual data [1]. As a rule, no study in machine learning is complete without experiments on real or model data confirming the practical performance of the method. There are various methods for solving this problem, but universal methods have not been found at the moment [2–4]. In addition, in the development of algorithms, the possibility of testing them is important [5]. This article will focus on a specialized software package that makes it possible to simultaneously create new data for training and test existing algorithms in simulation mode. The need to develop such software is due to the requirement to obtain related data for some tasks. We are discussing combined data sets of visible and infrared ranges obtained with a single camera or stereo pair, and additional information in the form of lidar data or simply a depth map, a segmentation picture, or data on the location of objects of interest in a photo or video image. A detailed analysis of various open sources showed that there are currently no such datasets [6]. The effort led to the creation of software that enables data synthesis to produce training examples. The developed descriptions of the internal structure of this software enable further improvement of the approaches and the possibility of modifying the resulting pipeline for different goals and objectives.

SIMULATOR OPERATION DESCRIPTION

The developed simulator is designed to solve problems associated with training and testing the movement of an unmanned robotic vehicle. The main functions of the simulator are:

– carrying out *software-in-the-loop* and *hardware-in-the-loop* testing of algorithms used in the equipment of robotic technical complexes (RTC) in "scaled real time" mode;

– generation of visible and infrared images of the scene, as well as reference markup, for the development and training of neural network algorithms for object recognition and semantic segmentation of the scene;

– debugging algorithms of stereo reconstruction, visual odometry and SLAM.

The simulator mimics such sources of sensor output data, including the effects of their imperfection, as television (TV) and thermal imaging (IR) cameras, lidars, satellite navigation systems, inertial systems (INS), wheel odometry of a ground wheeled robotic vehicle (WRV), sensor of the angle of rotation of the WRV wheels.

The presented software component supports several scenarios of operation.

Generator mode allows generating sets of related data to train algorithms. On a number of cluster GPUs, data is generated in parallel and saved in the cluster memory.

Test mode allows you to test individual pieces of a dataset in real time using a web interface.

The field test mode is a testing ground over which some physical model, such as an unmanned aerial vehicle or a car, can move. In this mode, the loop is closed by feedback, allowing you to test algorithms in conditions close to real, without the consequences of physical implementation.

Manual mode allows you to intercept the control of the physical model by the operator in order to make

corrections during tests. A program simulates a sequential video sequence and transfers data using network sockets to a demo computer with equipment, as well as to another cluster that recognizes simulated images and provides feedback. During testing of an unmanned vehicle, it is not necessary to involve a human operator for additional control over what is happening, since there is no danger of physical impact [7, 8], moreover, several scenarios can be launched simultaneously for quick hypothesis testing.

A diagram of the simulator is shown in Fig. 1.

The developed software is a universal implementation of the simulator-generator. The internal structure is a set of models, shader programs, and scripts that are interconnected into a single representation. The configuration *json* file is used to configure the modes of the software complex and the internal state of the scenes.

Fig. 1. Simulator diagram

In test mode, two local folders are kept in the configuration. The first contains the descriptions of frames in *json* format, and the second receives the files generated from these descriptions. Thanks to the test mode, it becomes possible to pre-view large scenes without wasting time on creating an extensive assembly.

The generator mode allows for extensive random sampling by randomizing the camera position and objects. The scene description is also in *json* format. Description is delivered to the software via a dedicated queue. Then a visual picture and mask for the location of objects of interest are drawn. Also, a description is made of a separate frame with the names of objects and their positions in pixel coordinates. Generating a large amount of data can take quite a while, but it is possible to parallelize tasks. There are 8 GPUs on the cluster, each of which is involved in generating a large sample. A web application is used to control the generator. In it, you can form a job, specify the number of GPUs to be involved. Sampling generation status, each generation step is sent to the web application. After completion of the generation process using the web application, it is possible to obtain the generated data via a direct link. The generator diagram is shown in Fig. 2.

Fig. 2. Generator diagram

A VIRTUAL OPTOELECTRONIC SYSTEM FOR EXPERIMENTS WITH ALGORITHMS

The optical electronic system (OES), used to conduct experiments with algorithms, is an ideally rigid assembly of 4 stereo pairs. The optical axes of cameras lie in the same plane. The azimuth angles of the optical axes of stereopairs are 0, 90, 180, 270°. The horizontal field of view of each stereo pair is 94°, which ensures the intersection of the fields of view. Note here that this value is typical. For the purpose of specific experiments, this value can be changed (for example, in order to optimize the algorithms). The vertical field of vision of each stereo pair is 70° (also typical but can vary). The distance between the optical axes of cameras that comprise stereo pair (base) is 1 m (may vary). The optoelectronic system lets you change scene parameters using input parameters. Camera spacing and camera parameters, such as focal length, optical sensor size, are adjusted using a configuration file.

The output format is eight images from cameras, location coordinates for the center of the car, and its rotation in the global coordinate system. The positions and rotations of the cameras in the local space of the car are also known. The output data is presented in the form of a folder structure containing scene descriptions in *json* format and images from cameras.

DESCRIPTION OF THE CONFIGURATION DATA FORMAT FOR THE GENERATOR

Data structure for the generator

The generator should obtain comprehensive information about the types of objects in the frame, the location of objects, lighting angles, the need to enter objects in the scene description, etc. Data is sent to the software in *json* format and has the following structure.

```
{
    "Count": 1,
    "Comment": "",
    "Map": "Test_Track_00001",
    "BackgroundObjects": [_],
    "Foreground Objects": [_]
    "Cameras": [_]
    "Images": [_]
    "Environment": {_}
    "NOPlacements": [_]
    "DOPlacements": [_]
    "Sensors": [_]
},
```

Description of `Count`, `Comment`, `Map`, `BackgroundObjects`, `Foreground Objects` fields in generator data

– The **`Count`** field stores the number of frames that the generator will need to make.

– The **`Comment`** field can include an explanation for the frame/set of frames.

– The **`Map`** field stores a string that corresponds by name to one of the static cards that were previously prepared and configured for use.

– The **`BackgroundObjects`** field stores an array of off-screen stationary objects which positions are nor intended to change, but can be used to alter the scene.

– **`Foreground Objects`** field is an array of strings with the names of the so-called foreground objects. Such objects are the "actors" of the scene, and usually cameras are installed on them. It can be a car with all its sensors, but in addition to a car, it can be a quadcopter or other drone filming a car from above.

Description of the `Images` field in the generator data

The **`Images`** field lists the frames that all cameras need to make on stage. This is usually a snapshot in the visible spectrum, infrared, and a snapshot of a segmentation camera that allows you to capture objects on the stage in the form of solid spots.

An example of the content of the **`Images`** field

```
"Images": [
    {
        "Tag": "image",
        "ImageType": "Visible",
        "Camera": 0
    },
    {
        "Tag": "mask",
        "ImageType": "Mask",
        "Camera": 0
    }
],
```

Description of the `Environment` field in generator data

The **`Environment`** field contains a structure that defines the lighting parameters on stage. The structure describes the behavior of the directional light. This does not mean that the scene cannot contain other types of lighting, but you need to add them in advance to the static component of the scene.

Scene structure in the **`Environment`** field

```
"Environment": {
    "LightIntensity": 1,
    "AmbientIntensity": 1,
    "ShadowPitch": 49,
    "ShadowRoll": -4,
    "ShadowIntensity": 1,
    "ShadowEnabled": true
},
```

Scene description fields define the lighting settings for the scene.

– The **`LightIntensity`** field is responsible for the intensity of the directional light.

– The **`AmbientIntensity`** field allows you to select the intensity of scattered ambient light.

– Directional light angles are set using **`ShadowPitch`** fields and **`ShadowRoll`**.

– **`ShadowIntensity`** field ranges from 0.0 up to 1.0 and controls the brightness of the shadow. This setting can also be affected by excessive **`AmbientIntensity`**.

– **`ShadowEnabled`** field allows you to enable and turn off shadows from the directional light.

Description of the `NOPlacements` field in generator data

The **`NOPlacements`** field is an array of structures describing moving objects on the scene.

Content of the **`NOPlacements`** field:

```
"NOPlacements": [
    {
        "Id": "fg0",
        "ObjectPlacement": {
            "PlacementType": "absolute",
            "ParentId": null,
            "Position": {
                "X": 14223.819710696813,
                "Y": -3410.0956920878875,
                "Z": 100,
                "Yaw": 90,
                "Pitch": 0,
                "Roll": 0
            },
            "Scale": {
                "ScaleX": 1,
                "ScaleY": 1,
                "ScaleZ": 1
            },
            "Model": {}
        }
    },
    -
],
```

– The structure has an **Id** field that serves as an alias for the specified object.

– The **ObjectPlacement** field contains a structure describing the placement of dynamic objects.

– The **PlacementType** field describes the placement method. It can be **absolute**, i.e., the position of the object in the global coordinate system is specified. The **PlacementType** field, set to **relative**, allows you to place objects relative to the specified object.

– The **ParentId** field allows you to set the parent object for the current object, the position in the **relative** mode will be calculated relative to the first one.

– The **Position** field is a structure that describes the position of the object. In this structure, the position parameters specified by the **X**, **Y**, **Z** fields are specified, and the rotation of the object is also recorded in Euler coordinates using the **Yaw**, **Pitch**, **Roll** fields specified in degrees.

– The **Scale** field contains a structure that describes the scale of a dynamic object along three axes using the **ScaleX**, **ScaleY**, **ScaleZ** parameters.

– The **Model** field contains a structure that allows you to specify which motion model an object should use in dynamics.

Description of the Cameras field in generator data

The **Cameras** field is an array that contains objects for setting each camera on the scene separately.

Camera setup appearance:

```
"Cameras": [
{
      "PixelSizeX": 3.45e-06,
      "PixelSizeY": 3.45e-06,
      "FocalLength": 0.01875,
      "MatrixW": 6144,
      "MatrixH": 6144,
      "CameraMainOffset": 0.0,
      "CameraCrossOffset": 0.0,
      "CameraAxisAngle": 0.0,
      "ImageFormat": "png",
      "ObjectId": "fg0",
      "CameraId": "forward_cam_0",
      "IsOrtho": false,
      "OrthoSize": 200
     "ImageEnhancementParameters": {
           "CalibrationRulerOn": false,
           "CalibrationRulerDistance": 0.0,
           "DownScaleSimulationOn": false,
           "FrameScaleX": 0.0,
           "FrameScaleY": 0.0,
           "CameraShakingOn": false,
           "ExPositionTimeSeconds": 0.0,
           "NonIdealEffectsOn": false
           },
     },

].
```

– In the **PixelSizeX** and **PixelSizeY** fields, set the pixel size in the virtual camera. In this case the standard pixel size of 3.45 μm, which corresponds to the real camera installed on the car, is set.

– The **FocalLength** field refers to the focal length of the camera, the real value corresponds to 0.01875 m.

– The **MatrixW** and **MatrixH** fields set the size of the output image in pixels.

– **CameraMainOffset**, **CameraCrossOffset** and **CameraAxisAngle** fields allow you to specify camera offsets in meters relative to the shooting center, as well as rotation.

– The **ImageFormat** field allows you to get the desired image format.

– The **ObjectId** field allows you to pin the camera to an assigned object.

– The **CameraId** field allows you to assign a unique identifier to the camera, in the future, you can access the camera with the identifier specified in this field.

– The **IsOrtho** field allows you to switch the camera to orthonormal mode, when this field is turned on, you also need to specify the settings for the orthonormal view, for example **OrthoSize** — the size of the orthonormal area.

– The **ImageEnhancementParameters** field stores a structure that allows you to describe the post-processing of a frame. You can add a calibration grid to a frame by activating the **CalibrationRulerOn** field and, if SerDe is enabled, you can specify the distance from the camera in the **CalibrationRulerDistance** field.

– **DownScaleSimulationOn** field enables high-resolution frame rendering mode and scaling to the dimensions specified in the **MatrixW** and **MatrixH** fields using a pre-specified compression algorithm.

– Compression can also be set manually using the **FrameScaleX** and **FrameScaleY** fields.

– The **CameraShakingOn** field activates the image blur mode when camera is shaking. In this mode, several frames are drawn at once, which are summed up according to some predetermined algorithm. For this mode, you also need to set the exposure time using the **ExPositionTimeSeconds** field.

– The **`NonIdealEffectsOn`** field includes a post-processing mode that uses a pre-selected algorithm to add camera-specific defects to the frame.

`Sensors` field description in generator data

The **`Sensors`** field is used to provide information about additional testing devices. The description, implementation, and interaction of the device vary depending on the implementation of the generator. When testing equipment is included in the **`Sensors`** list and is available, the generator incorporates sensor data into the accompanying information for the generated data set. In the implementation of the generator for an unmanned vehicle, there are lidar sensors, rear and front radar sensors, a wheel odometry sensor, a gyroscopic sensor, and an accelerometer.

DATA SIMULATION MODE

The generator is used to create a data set within each study at a time, which takes a relatively long time. In the future, as part of this study, the start of the generator may not be required.

As a stage for experiments with algorithms, a virtual city with the surrounding countryside is used. The stage consists of several dozen buildings of different heights, together forming a city block, and an adjacent rural area with green spaces. There are intersections, roundabouts, pedestrian crossings, traffic signs, parking spaces, and road sections under repair. The city is subject to the movement of cars and pedestrians. An example of the urban environment image generated using the developed photorealistic simulator is shown in Fig. 3.

Fig. 3. An example of an image of an urban environment generated using the developed photo-realistic simulator

At the same time, the simulator should be designed for frequent use, so the simulator mode imposes increased requirements on the frame generation time. Scenes intended for the simulator should be optimized, and post-processing of frames should be reduced to a minimum. Optimization also concerns sensor data.

To assess the accuracy of the algorithms, the metric can be calculated using a segmentation mask, but mask generation is often not advisable for simulation. For evaluation metrics, less expensive methods are used, for example, generating markup using a bounding box projection.

In addition to the maximum optimization of the scene and models, it is also necessary to optimize the exchange protocol. In the developed software, in order to save more bandwidth on network channels, it was decided to compress the images generated by the simulator. Most of the information is collected using cameras located on the roof of the car. Eight cameras are supposed to be used, set up in such a way that the cameras capture the field of view of their neighbors. Such a construction was chosen for the subsequent stacking of images into a panorama. In addition, the simulator has functionality to get a finished panoramic image for further processing by neural network algorithms. An example of a panoramic image is shown in Fig. 4.

Fig. 4. An example of a panoramic shot

CONCLUSION

A description of software for creating synthesized data and a feedback simulator for testing machine learning algorithms are presented. The developed simulator is designed to solve problems associated with training and testing the movement of an unmanned robotic vehicle, and supports data generator operation modes, test mode, field test mode, and manual mode. The structure of the developed software is described.

The presented software allows testing an unmanned vehicle without the involvement of a human operator, i.e., there is no danger of physical impact. It is possible to run several scenarios at the same time to speed up the testing of hypotheses.

REFERENCES

1. Medvedev M., Kadhim A., Brosalin D. Developmentof the Neural-Based Navigation System for a Ground-Based Mobile Robot. *7th International Conference on Mechatronics and Robotics Engineering, ICMRE 2021,* 2021, pp. 35–40. DOI: 10.1109/ICMRE51691.2021.9384825
2. Kaftannikov I.L., Parasich A.V. [Problems of training set's formation in machine learning tasks]. *Vestnik Yur-GU. Seriya "Komp'yuternye tekhnologii, upravlenie, radioehlektronika"* [Bulletin of the South Ural State University. Series: Computer Technology, Automatic Control, Radio Electronics], 2016, vol. 16, no. 3, pp. 15–24. DOI: 10.14529/ctcr160302 (In Russ.).
3. Roh Y., Heo G., Whang S.E. A survey on data collection for machine learning: a big data AI integration perspective. *IEEE Transactions on Knowledge and Data Engineering*, 2021, vol. 33, is. 4, pp. 1328–1347. DOI: 10.1109/TKDE.2019.2946162
4. Gavrilov D.A., Schelkunov N.N. [Software for large format aerospace image marking and training samples prepa-

ration]. *Nauchnoe Priborostroenie* [Scientific Instrumentation], 2020, vol. 30, no. 2, pp. 67–75. DOI: 10.18358/np-30-2-i6775 (In Russ.).
5. Gavrilov D.A. [The computer system testing of algorithms for detection and localization of objects in video sequences]. *Nauchnoe Priborostroenie* [Scientific Instrumentation], 2019, vol. 29, no. 1, pp. 149–156. DOI: 10.18358/np-29-1-i149156 (In Russ.).
6. Lapushkin A.G., Gavrilov D.A., Shchelkunov N.N., Bakeev R.N. [The main approaches to the preparation of visual data for training neural network algorithms]. *Iskusstvennyi intellekt i prinyatie reshenii* [Artificial intelligence and decision making], 2021, no. 4, pp. 62–74. DOI: 10.14357/20718594210406 (In Russ.).
7. Pisareva O.M., Alexeev V.A., Mednikov D.N., Starikovsky A.V. [Characteristics of vulnerability zones and threats sources for information security by the operation of unmanned vehicles in an intelligent transport system]. *Nauchno-tekhnicheskie vedomosti SPbGPU. Ehkonomicheskie nauki* [St. Petersburg State Polytechnical University journal. Economics], 2021, vol. 14, no. 4, pp. 20–36. DOI: 10.18721/JE.14402 (In Russ.).
8. Li L., Huang W.-L., Liu Y., Zheng N.-N., Wang F.-Y. Intelligence testing for autonomous s vehicles: a new approach. *IEEE Transactions on Intelligent Vehicles*, 2016, vol. 1, is. 2, pp. 158–166. DOI: 10.1109/TIV.2016.2608003

Contacts: *Lapushkin Andrey Georgievich,*
lapushkin.ag@mipt.ru

Article received by the editorial office on 31.10.2022