УДК 004.896:004.932:004.421.2

ПОСТРОЕНИЕ СЕГМЕНТИРОВАННОЙ ЛОКАЛЬНОЙ КАРТЫ ДЛЯ ПЛАНИРОВАНИЯ МАРШРУТА ДВИЖЕНИЯ БЕСПИЛОТНЫХ НАЗЕМНЫХ ТРАНСПОРТНЫХ СРЕДСТВ НА ОСНОВЕ СИСТЕМЫ ДВУХМЕРНОГО КРУГОВОГО ОБЗОРА

А. В. Молчанов[1]

В работе рассматривается метод картирования окружающего пространства вокруг беспилотного наземного транспортного средства, основанный на системе двухмерного кругового обзора, работающей за счёт четырёх предварительно откалиброванных стереокамер и объединения изображений с них в одно – так называемую локальную карту. Дальнейшее применение инструментов машинного и глубокого обучений позволяет выделить на карте проходимые и непроходимые области и использовать эту информацию для осуществления автономной навигации.

Введение

Одним из элементов или даже концепцией при разработке беспилотных наземных транспортных средств (далее – *БНТС*) является их восприятие окружающего пространства, которое служит основой для безопасного и эффективного передвижения подобного рода транспортных средств как по просторным и безлюдным складским помещениям, так и по тесным городским улицам, переполненным людьми.

Для решения этой задачи используются различные типы сенсоров, включая *радары*, *видеокамеры* и *лидары*, однако традиционные методы картирования зачастую требуют значительных вычислительных ресурсов, а также не всегда обеспечивают достаточный уровень детализации в режиме реального времени [1], что является критической проблемой для любого вида беспилотного транспорта, управляет которым большую часть времени не человек, а компьютер.

1 *432027, Ульяновск, ул. Северный Венец, 32, УлГТУ, e-mail: ydx-ayrtom@yandex.ru*

Одним из возможных решений данной проблемы является использование *системы двухмерного кругового обзора*, которая может работать за счёт разного количества камер и позволяет получать довольно точные и детализированные данные о пространственном окружении транспортного средства без необходимости сложной и дорогостоящей во всех отношениях обработки *трёхмерных облаков точек* с лидара.

Однако при таком подходе ключевой проблемой является *сегментация* полученных данных и выделение на них значимых объектов, таких как препятствия, люди, дорожные знаки с разметками и зоны, пригодные для передвижения. В данном случае, без надёжного и устойчивого алгоритма сегментации невозможно построение корректной *сегментированной локальной карты* на основе системы двухмерного кругового обзора, что, в свою очередь, затрудняет эффективное, а что самое главное безопасное планирование маршрута движения БНТС.

Актуальность этого исследования обусловлена растущими требованиями к автономным транспортным системам, нуждающимся в достаточно точных и вычислительно эффективных алгоритмах восприятия окружающей их среды для повышения безопасности и эффективности транспортной инфраструктуры подобного рода.

Робототехнический 3D-симулятор *Webots* [2] и Python-фреймворк *Robot Operating System* (ROS 2) [3] было решено выбрать для разработки и тестирования всего решения в целом из-за их доступности, технических возможностей и активного сообщества.

Калибровка виртуальных видеокамер

Поскольку система двухмерного кругового обзора, устройство которой будет более подробно описано в следующей главе, полностью завязана на данные, поступающие от четырёх отдельных виртуальных *стереокамер*, то правильно было бы, перед началом какой-либо обработки этих данных, устранить в них любые возможные искажения, которые, применительно как к электронным оптическим системам, каковыми и являются большинство современных камер, так и к их продукту – RGB-изображениями, могут иметь следующую природу:

1. *Радиальная дисторсия*, которая изгибает прямые линии на изображении от («подушкообразная») или к («бочкообразная») его центру и задаётся следующими формулами по модели *Брауна-Конради* [4]:

$$x_{distorted} = x\left(1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6\right) \quad (1)$$

$$y_{distorted} = y\left(1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6\right) \quad (2)$$

где: $x_{distorted}, y_{distorted}$ – координаты с учётом искажения; *x, y* – нормализованные координаты точки (в системе координат камеры, где *z* = 1); *r* – расстояние от *оптического центра* камеры (чем дальше точка от центра изображения, тем сильнее искажения); $k_1 - k_3$ – нелинейные коэффициенты радиальной дисторсии;

2. *Тангенциальная дисторсия*, которая наклоняет вертикальные и горизонтальные линии на изображении, искажая таким образом перспективу. Задаётся следующими формулами по модели Брауна-Конради:

$$x_{distorted} = x + \left[2 p_{1\,xy} + p_2\left(r^2 + 2x^2\right)\right] \quad (3)$$

$$y_{distorted} = y + \left[p_1\left(r^2 + 2y^2\right) + 2 p_2 xy\right] \quad (4)$$

где: p_1, p_2 – нелинейные коэффициенты тангенциальной дисторсии;

3. Другие виды и типы искажений [5], которые ввиду идеальных условий и данных в симуляторе не будут эмулироваться в данной работе.

На Рисунке 1 ниже представлена сцена для калибровки виртуальных видеокамер. В левой части изображения можно наблюдать дерево характеристик (горизонтальный угол обзора, ширина и высота изображения, искажения объектива, *фокусное расстояние*, оптический центр и др.) узла под названием «*camera_front_left*», который представляет собой объект стереокамеры, расположенной в левой части 3D-модели беспилотного автомобиля, и из которого исходят три оси X, Y и Z (см. Рис. 1). Расположение оставшихся трёх камер, а также углы их обзора можно определить по усечённой форме камеры (англ. *camera frustum*) в виде пирамид.

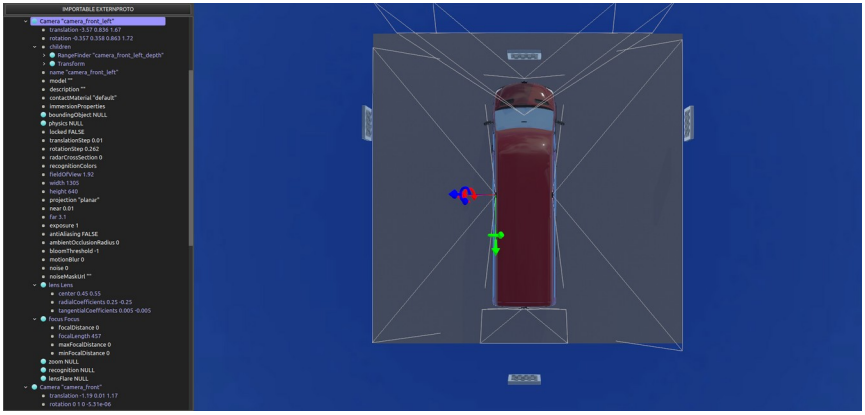

Рис. 1. Сцена №1 в Webots

Эмуляция конкретной модели используемых камер

Реальная конкретная модель камер, используемая в решении – *ZED 2* от *Stereolabs* [6], которая обладает следующими ключевыми для данной работы характеристиками:

Таблица 1. Характеристики реальной стереокамеры модели ZED 2

Характеристика	**Значение**
Горизонтальный угол обзора	110°
Вертикальный угол обзора	70°
Выходное разрешение	2 x (1920x1080) @ 30 FPS 2 x (1280x720) @ 60 FPS
Фокусное расстояние	528 пикселей (HD720)
Минимальное значение глубины	0.3 м
Максимальное значение глубины	20 м

Для их симуляции и быстрого применения одновременно к нескольким узлам объектов видеокамер в Webots, был написан специальный Python-скрипт вместе с конфигурационным файлом для него, представленном в следующем Листинге 1:

```
# DEF-наименования объектов типа Camera в Webots
devices_name: [
'camera_front_left_depth',
'camera_front_left',

'camera_front_depth',
'camera_front',

'camera_front_right_depth',
'camera_front_right',

'camera_rear_depth',
'camera_rear',
]

# Учитывать значения горизонтального и вертикального углов обзора
use_fov_values: True

horizontal_fov: 110 # Градусы
vertical_fov: 70    #

image_width: 1305 # Пиксели

# Пересчитываются при use_fov_values: True
image_height: 640 # Пиксели
focal_length: 457 #

# Упрощённая модель искажения Брауна-Конради
distortion_center: [0.45, 0.55]         # 0.0 ≤ x, y ≤ 1.0
radial_distortion: [0.25, -0.25]        # k1, k2
tangential_distortion: [0.005, -0.005] # p1, p2

# Для стереокамер с объектом типа RangeFinder в Webots
min_range: 0.3  # Метры
max_range: 20.0 #
```

Листинг 1. Характеристики виртуальной стереокамеры модели ZED 2

Исполнение скрипта перезаписывает файл с описанием Webots-сцены, применяя таким образом указанные в конфиге параметры к узлам из списка *devices_name* (см. Лист. 1). Такой подход позволяет не только сократить время на задание всех этих параметров вручную для того, чтобы добиться желаемого результата, но и предоставляет возможность сэмулировать конкретную модель камеры, которую планируется использовать, а также внести в её идеальные данные описанные ранее искажения объектива и перспективы по «упрощённой» модели Брауна-Конради. Упрощённая она по тому, что в Webots (см. Рис. 2) отсутствует третий коэффициент k_3 радиальной дисторсии:

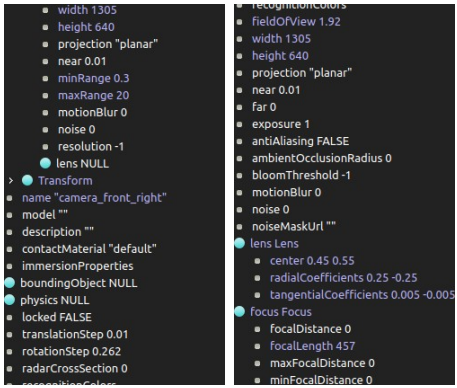

Рис. 2. Применённые параметры правой виртуальной стереокамеры модели ZED 2

Напротив, и в области обзора каждого из статичных устройств расположены шахматные доски (см. Рис. 1), которые начинают двигаться относительно них при запуске симуляции. В это время видеокамеры фиксируют *паттерны калибровки* в различных положениях и на основе известных размеров и геометрических преобразований подбирают свои внутренние параметры (фокусное расстояние, оптический центр и описанные выше коэффициенты дисторсий), а также учитывают их впоследствии при устранении искажений на новых изображениях. Так работает «*Новая гибкая технология калибровки камеры*» [7], теоретическая основа которой была предложена ещё в 1999 году китайско-американским учёным в области компьютерного зрения *Цзэнцзю Чжаном*.

В данной работе, для калибровки виртуальных видеокамер была взята её готовая реализация из самой популярной и открытой библиотеки компьютерного зрения *OpenCV* [8]. В Листинге 2 и на Рисунке 3 ниже представлен конечный результат работы калибровочного модуля.

```
%YAML:1.0
---
image_resolution: !!opencv-matrix
...
data: [ 640, 1305, 4 ] # image_height, image_width, channels (depth)
camera_matrix: !!opencv-matrix
...
# fx, 0., cx, 0., fy, cy, ...
data: [ 455.40140577414064, 0., 652., 0.,
455.40140577414064, 319.5,
0., 0., 1. ]
distortion_coefficients: !!opencv-matrix
...
# k1, k2, p1, p2, k3
data: [ 0.094175197313633899, -0.029941689611480013,
-0.0019921223506874112, 0.0028732521915666043, 0. ]
```

Листинг 2. Подобранные OpenCV параметры одной из виртуальных камер

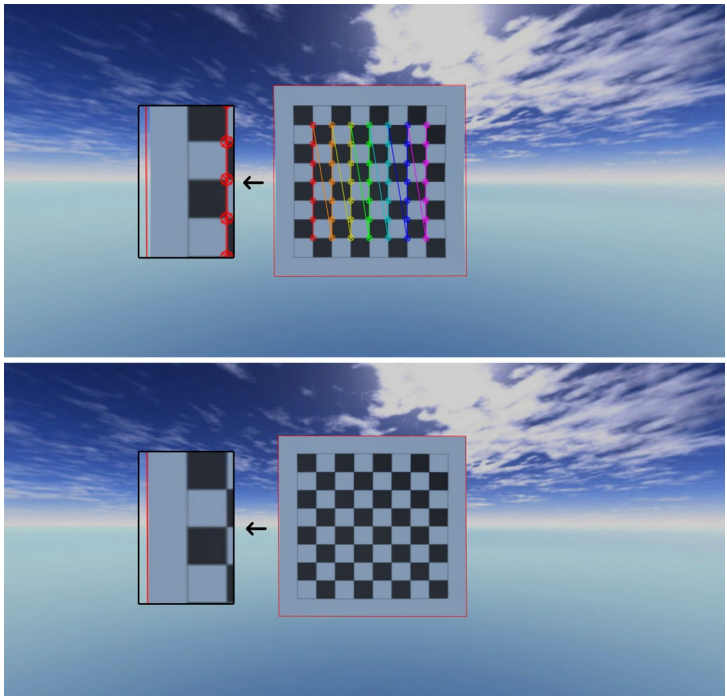

Рис. 3. Искажённое (сверху) и результирующее (снизу) изображения

Система двухмерного кругового обзора

На Рисунке 4 ниже представлен вид сверху на сцену для формирования системы двухмерного кругового обзора, а на следующем за ним Рисунке 5 – итоговый результат. Виртуальные размеры напольной шахматной доски – 20x25 метров, её одной клетки – 1.25 метра, квадрата с четыремя окружностями внутри – 5 метров.

Вид, представленный на Рисунке 5 называют *видом с высоты птичьего полёта* (англ. *Bird's-Eye View – BEV*). Визуальный эффект вытягивания высоких объектов на нём, например, бочек (см. Рис. 5), возникает от того, что применяемое в системе *гомографическое преобразование* корректно работает только для низких объектов, лежащих на одной плоскости с нулевым уровнем (землёй), в то время как высокие объекты выступают над этой плоскостью. При проецировании такая высота не учитывается и объекты как бы отбрасываются на землю, но со смещением от реальной точки основания.

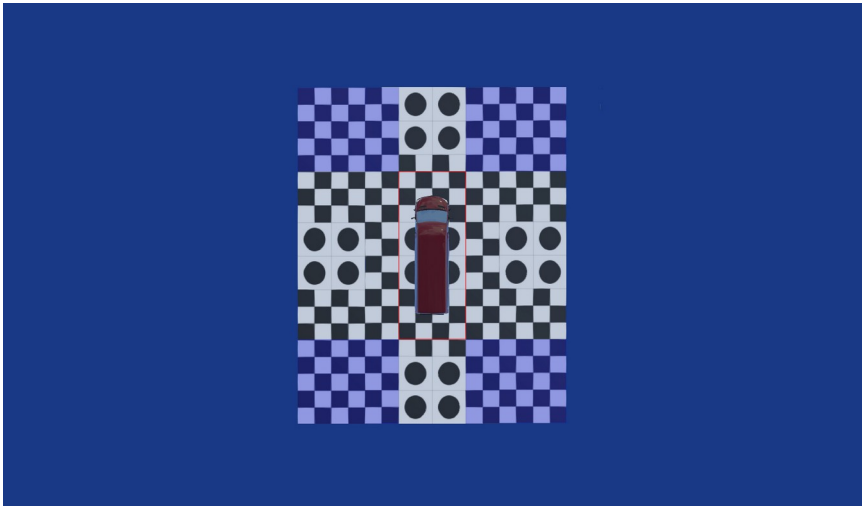

Рис. 4. Сцена №2 в Webots

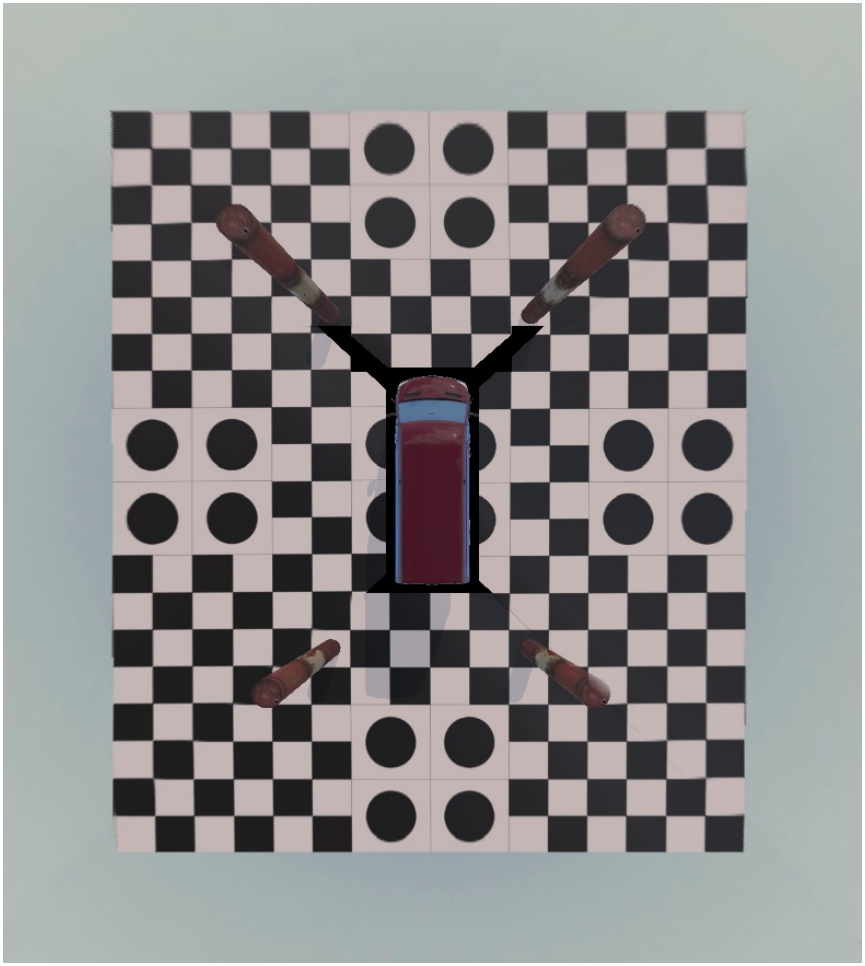

Рис. 5. Итоговый результат работы системы двухмерного кругового обзора

Формирование же самого единого изображения из четырёх отдельных осуществляется по следующему алгоритму:

1. Видеокамеры попарно располагаются таким образом, чтобы каждые две смежные из них хотя бы частично захватывали одну из угловых областей, помеченных на Рисунке 4 выше. Так, например, для левой и передней камеры это будет левая верхняя область, а для задней и правой – нижняя правая. Обводка вокруг 3D-модели автомобиля на всё том же Рисунке 4 – это «слепая» зона, которая не покрывается обзором камер;
2. Для каждого из устройств, по четырём ключевым точкам выбираемым вручную, рассчитывается так называемая *матрица проекции*, которая преобразует исходное перспективное изображение (верхнее окно на Рисунке 6 ниже) в него же, но с видом сверху:

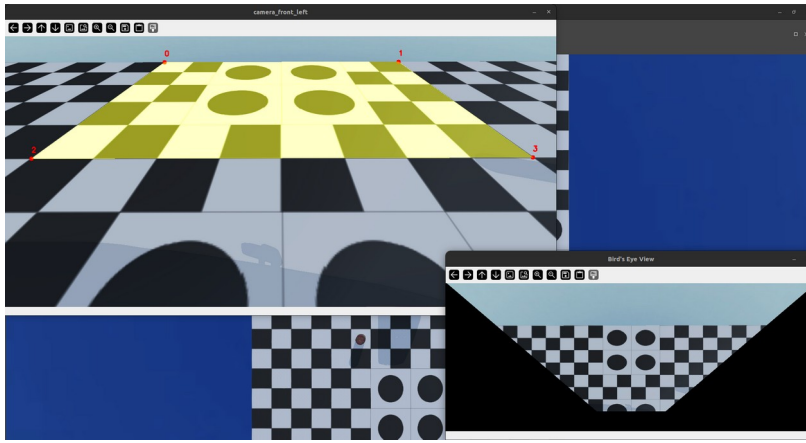

Рис. 6. Результат применения проекционной матрицы (нижнее окно)

3. Затем для каждой пары смежных видеокамер на этих изображениях находится та самая перекрывающаяся область через вызов методов *cv2.bitwise_and* и *cv2.bitwise_not*, после чего из неё создаётся *бинарная маска* через вызов методов *cv2.threshold* и *cv2.dilate*. На Рисунке 7 ниже и далее до конца данной главы в качестве примеров будут представлены сторонние изображения из документации к оригинальной работе [9], на которой была основана описываемая уже здесь система двухмерного кругового обзора:

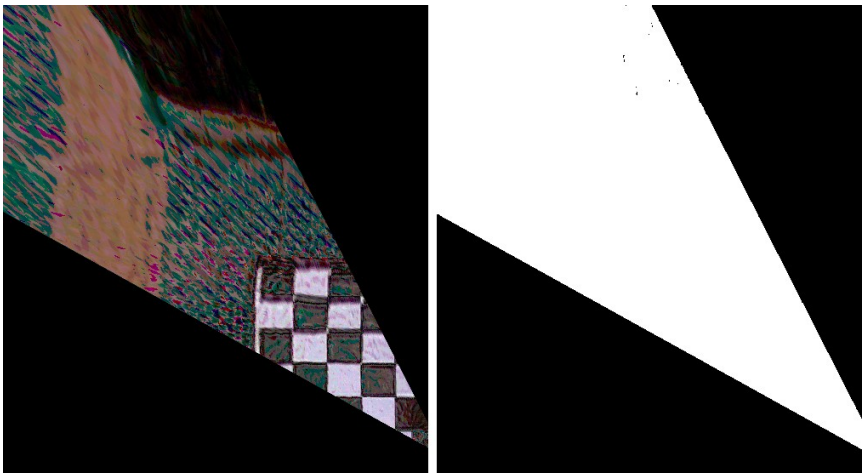

Рис. 7. Пример перекрывающейся области (слева) и её бинарной маски (справа)

4. Через применение методов *cv2.findContours* и *cv2.approxPolyDP* на всё тех же изображениях находятся контуры за пределами перекрывающейся области и относятся к той или иной камере.
   Для каждого пикселя в общей части двух изображений рассчитывается расстояние до двух найденных ранее полигонов через вызов метода *cv2.pointPolygonTest* и его вес по следующей формуле:

$$w = d_B^2 / \left(d_A^2 + d_B^2 \right) \quad (5)$$

   где: d_A, d_B – расстояния до полигона *A* (например, передняя камера) и полигона *B* (например, левая камера). Если пиксель составляет переднее изображение, то расстояние от него до полигона B будет больше, увеличивая таким образом общее значение веса. Пиксели за пределами общей части будут иметь веса равные 0 для левой камеры и 1 для передней;
5. Используя описанную выше информацию о весах каждого пикселя в перекрывающейся и неперекрывающейся областях изображений с любых двух смежных камер, составляется *весовая матрица G*, значения которой лежат в диапазоне от 0 до 1 и которая представлена на следующем Рисунке 8 вместе с контурами (полигонами), упомянутыми ранее:

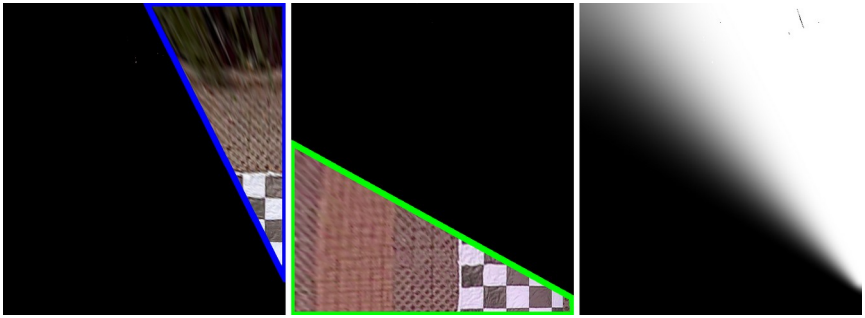

Рис. 8. Полигон A, полигон B и весовая матрица G

Объединённое итоговое изображение с видом сверху для левой и передней видеокамер может быть получено как:

$$fusedImage = frontImage * G + (1 - G) * leftImage \quad (6)$$

где: *frontImage, leftImage* – исходные изображения с соответствующих камер после применения к ним матриц проекции (см. Рис. 6).

По аналогии шаги 3-6 проделываются для переднего и правого, правого и заднего, заднего и левого устройств, что на выходе даёт четыре весовые матрицы *G0-G3*, которые вместе с четыремя масками перекрывающихся областей *M0-M3* объединяются в два единых изображения формата *RGBA* (см. Рис. 9) для дальнейшей репродукции итогового результата:

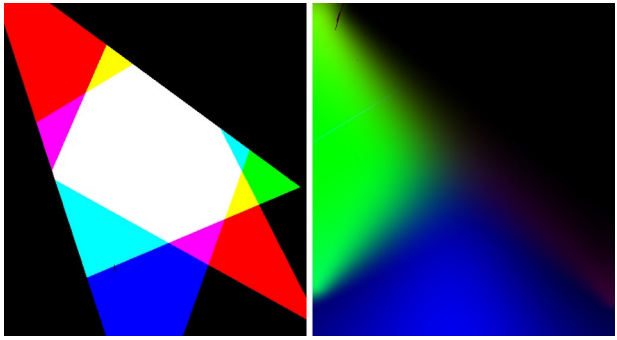

Рис. 9. Объединённые маски M и веса G

6. В заключение и при необходимости, на итоговом изображении двухмерного кругового обзора (см. Рис. 5) производится устранение различий в яркости и цветовом балансе.

Сегментированная локальная карта

На Рисунке 10 ниже представлена сцена для отладки и тестирования сегментированной локальной карты, а также алгоритмов планирования маршрута движения БНТС. Данный мир представляет из себя испытательный полигон с препятствиями в виде пластиковых бочек.

Задача *эго-автомобиля* в режиме автономного управления доехать до конца полигона, развернуться с использованием задней передачи в свободной от препятствий зоне и вернуться обратно, попутно со всем этим объезжая бочки. Сигналы светофора и знаки дорожного движения при этом никак не учитываются, однако, если нужно, могут быть использованы для проверки работоспособности и точности прочих алгоритмов.

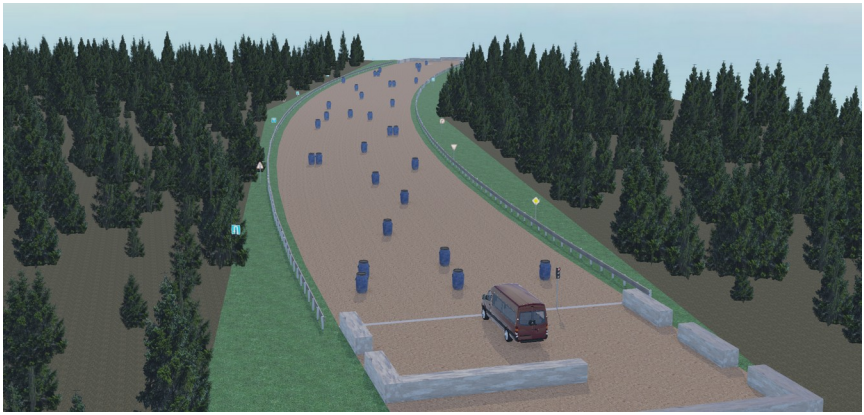

Рис. 10. Сцена №3 в Webots

FastSeg и YOLO11

Сама по себе, построенная локальная карта (см. Рис. 11 ниже) бесполезна, если речь идёт именно о беспилотном наземном транспортном средстве, потому что компьютер не может просто взять и поехать по ней, как это может сделать человек, – его нужно этому научить.

FastSeg – это легковесная модель сегментации, предназначенная для быстрого выделения объектов на изображениях и основанная на архитектуре типа *encoder-decoder* [10].

В контексте данного проекта, FastSeg используется для выделения проходимых (условная дорога) и непроходимых (всё остальное кроме дороги) для автомобиля областей на изображениях с камер кругового обзора, а также препятствий (см. Рис. 12), что является ключевым шагом при построении сегментированной локальной карты.

Рис. 11. Вид сверху в симуляторе (слева) и построенная локальная карта (справа)

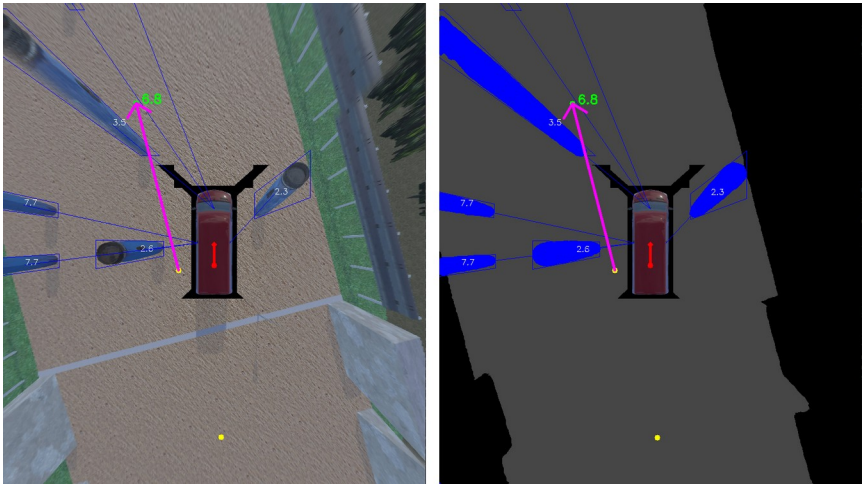

Рис. 12. Обычная (слева) и сегментированная (справа) локальная карта

Во введении данной статьи было упомянуто, что «без надёжного и устойчивого алгоритма сегментации невозможно построение корректной сегментированной локальной карты на основе системы двухмерного кругового обзора, что, в свою очередь, затрудняет эффективное, а что самое главное безопасное планирование маршрута движения БНТС».

Использование такой модели *глубокого обучения*, как FastSeg, решает обе эти проблемы, поскольку *свёрточные нейронные сети*, каковой она и является, в задачах сегментации и *детекции* уже довольно давно превзошли классические подходы *машинного обучения* как по надёжности, так и по устойчивости [11].

Другая серьёзная проблема, с которой учёные и разработчики борются теперь – это данные, их качество, а с недавних пор ещё и количество [12]. В этой работе, данную проблему было решено обойти стороной и собрать небольшое количество (100 изображений с каждой из используемых стереокамер) только тех данных, которые обе модели будут «видеть» на постоянной основе. Иными словами, FastSeg и *YOLO11* (о ней ниже) были намеренно переобучены и делали свои предсказания только на тех изображениях на которых и обучались. Такой подход позволил сосредоточиться на разработке, отладке и тестировании алгоритмов, а не на комплексном, трудоёмком и как следствие затратном по времени процессе сборки и формирования достаточно репрезентативных датасетов.

YOLO – это целое семейство моделей глубокого обучения для детекции объектов на изображениях и видео в режиме реального времени [13]. Основной принцип работы модели любой версии (в том числе и одиннадцатой) из семейства заключается в однократном проходе изображения через свёрточную нейронную сеть, что обеспечивает высокую скорость обнаружения объектов при приемлемой точности.

В контексте данного проекта, *yolo11n* (n – nano – самая легковесная модель) используется для распознавания препятствий вокруг эго-автомобиля, что позволяет учитывать их при сегментации локальной карты (см. тонкие линии до бочек в прямоугольниках на Рис. 12) и планировании маршрута движения.

Несмотря на то, что FastSeg уже и так был обучен сегментировать пластиковые бочки (см. Рис. 12 выше), это работает медленнее и плохо поддаётся масштабированию на другие виды препятствий из-за дополнительных временных затрат на их отдельное выделение (см. Рис. 13) в процессе разметки собранных данных, которая, к слову, производилась в веб-приложении *CVAT* [14].

Обучение обеих моделей производилось с использованием бесплатных мощностей платформы *Kaggle* [15] и её сервиса под названием «*Kaggle Notebook*».

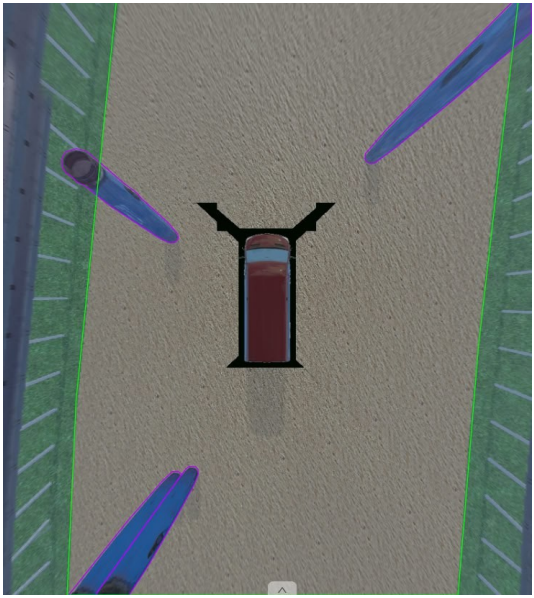

Рис. 13. Пример кадра из набора обучающих данных для модели FastSeg

Планирование маршрута движения БНТС

Сразу стоит отметить, что на момент написания данной статьи, какие-либо алгоритмы планирования маршрута движения БНТС не были интегрированы и не обеспечивают его автономное движение по испытательному полигону в Webots с использованием сегментированной локальной карты, а стрелки и дистанция (крупные цифры) до текущей точки пути на карте (см. Рис. 12), это лишь визуализация заранее записанных виртуальных GPS-координат и движение по ним. Мелкие цифры возле центра каждой из пластиковых бочек – это визуализация информации о расстоянии от виртуальных стереокамер до каждого из препятствий, которая берётся из *карт глубины* (см. Рис. 14), по одной с каждой камеры:

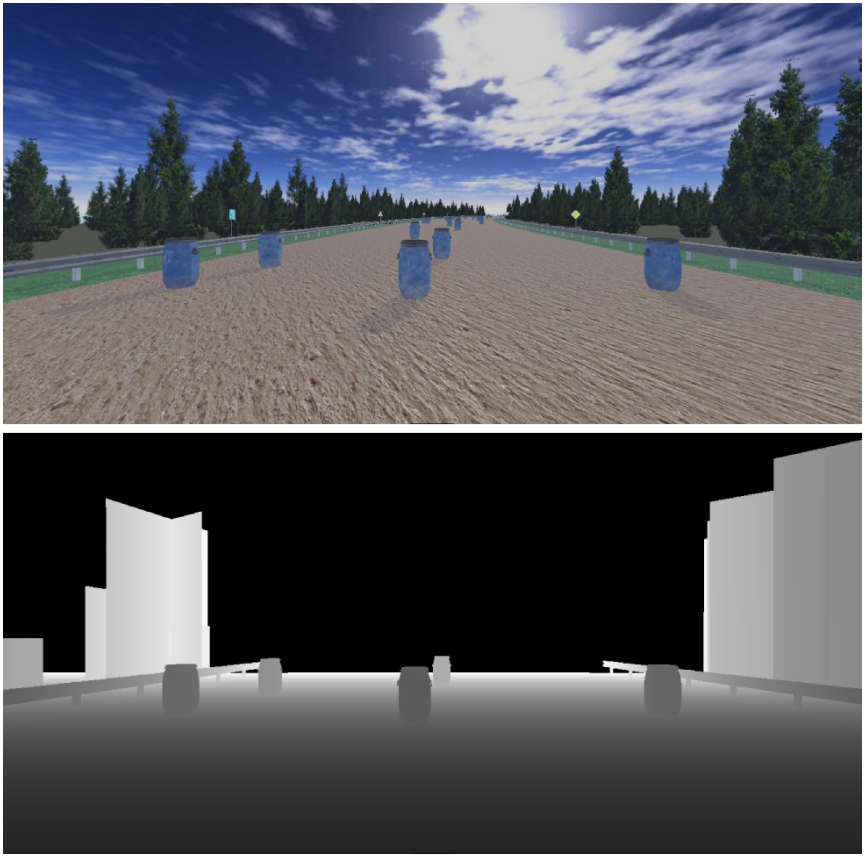

Рис. 14. Карта глубины (снизу) с передней стереокамеры в Webots

Несмотря на это, готовая реализация одного глобального и двух локальных планировщиков маршрута движения *Mercedes-Benz Sprinter* всё же присутствует в решении, однако в том его модуле, который не будет затронут в данной статье. Он включает в себя библиотеку для ROS 2 под названием *SLAM Toolbox* [16], которая предназначена для построения *глобальной карты*, а также фреймворк *Nav2* [17], предназначенный для автономной навигации и содержащий в себе упомянутые ранее алгоритмы, описание которых представлено далее.

NavFn Planner

NavFn Planner (Navigation Function Planner, далее – NFP) [18] является классическим *глобальным планировщиком* на 2D-сетке формата *OccupancyGrid* и *Costmap* (см. левую часть на Рис. 15 ниже), в случае с Nav2.

Основа NFP по умолчанию – это реализация одного из вариантов алгоритма *Dijkstra* для поиска кратчайшего, но грубого пути от текущей позиции робота на глобальной карте до цели, причём без учёта его динамики и кинематики. Именно поэтому для этих целей он обычно используется вкупе с *локальным планировщиком*, который уже и обеспечивает движение по построенному NFP маршруту с учётом ограничений платформы. Описание двух из таких локальных планировщиков можно найти ниже по тексту.

С математической точки зрения, Navigation Function Planner с Dijkstra «под капотом» работает следующим образом:

1. Создаётся граф $G=(V,E)$, где: V (вершины графа) – все проходимые для робота ячейки глобальной карты, которые имеют стоимость 0 и, следовательно, не содержат в себе препятствия (ячейки со стоимостью, например, 100 и более); E – рёбра графа между соседними ячейками (4 прямых и 4 диагональных). Через такой параметр данного планировщика как *allow_unknown* (True по умолчанию), можно добавлять в V ячейки карты с неизвестной стоимостью, в которых по тем или иным причинам непонятно что находится – область, по которой можно проехать, или препятствие;
2. Вес w ребра между двумя соседними вершинами (клетками сетки) $u=(x_1,y_1)$ и $v=(x_2,y_2)$ задаётся как:

$$w(u,v)=d(u,v)*(1+\alpha*c(v)) \quad (7)$$

где: $d(u,v)$ – евклидово расстояние между ячейками (1 – для прямых, $\sqrt{2}$ – для диагональных); 1 – базовый множитель чистого движения без препятствий; α – коэффициент, который определяет насколько сильно робот будет избегать потенциально опасных ячеек с высокой стоимостью. Регулируется через такой параметр фреймворка как *cost_scaling_factor* (10.0 по умолчанию); $c(v)$ – стоимость клетки из Costmap (см. Рис. 15), в которую планируется переход;

3. Все узлы $v \in V$ графа G добавляются в приоритетную очередь Q в порядке возрастания значения $g(v)$, которое формируется для каждой из свободных ячеек глобальной карты по следующей логике:

$g(s)=0$ – накопленная стоимость перехода от старта s до него же (не путать со стоимостью ячейки и со стоимостью перехода из текущей ячейки в соседнюю!),

$g(v)=\infty$ – начальная накопленная стоимость перехода от старта s до всех остальных ячеек;

4. Пока Q не пуста, извлекаем из неё вершину u с минимальной $g(u)$, которая находится ближе всего к старту s;
5. Рассматриваем все смежные (соседние) и проходимые вершины $v \in adj(u)$, в которые можно попасть из u за один шаг;
6. Считаем новую стоимость gn перехода из u в v:

$$gn=g(u)+w(u,v) \quad (8)$$

где: $g(u)$ – накопленная стоимость перехода от старта s до текущей ячейки u с учётом всех предыдущих шагов; $w(u,v)$ – стоимость перехода из текущей ячейки u в соседнюю к ней проходимую ячейку v **С** учётом стоимости второй из Costmap (не путать со стоимостью ячейки и с накопленной стоимостью всего пути!);

7. Если gn (новая стоимость всего маршрута от старта s до следующей в нём точки v) меньше $g(v)$ (ранее посчитанной стоимости всего маршрута от старта s до следующей в нём точки v), то:

1) в Q заменяем $g(v)$ на gn, поскольку теперь мы можем добраться до v за меньшую стоимость,

2) запоминаем, что предок v – это u, чтобы по достижению целевой точки всего глобального пути можно было добраться от неё до старта s и построить конечный вариант наиболее оптимального пути,

3) поскольку значение $g(v)$ в Q теперь стало меньше, обновляем приоритет очереди для v, показывая тем самым, что теперь от старта s до точки v можно добраться по более короткой траектории;

8. Возвращаемся к пункту 4.

Альтернатива Dijkstra при использовании NavFn Planner – это алгоритм *A**, переключение на который осуществляется через такой параметр данного планировщика как *use_astar* (False по умолчанию).

С математической точки зрения, Navigation Function Planner с A* «под капотом» работает следующим образом:

1. Аналогично Dijkstra, создаётся граф $G=(V,E)$ (см. пункт 1 в предыдущем списке);
2. Вес w ребра между двумя соседними вершинами (клетками сетки) $u=(x_1,y_1)$ и $v=(x_2,y_2)$ задаётся как простое *евклидово расстояние* $d(u,v)$ между ними (1 – для прямых, $\sqrt{2}$ – для диагональных), без учёта штрафа $(1+\alpha * c(v))$, как это делается в Dijkstra, потому что если сильно увеличить вес ребра за счёт него, но не учесть этого в *эвристике* $h(v)$ (см. пункт 3), то она может перестать обладать двумя ключевыми свойствами – допустимостью и монотонностью, первое из которых гарантирует, что A* найдёт оптимальный путь за счёт того, что эвристика никогда не будет переоценивать реальную минимальную стоимость пути $h_{truth}(v)$ от текущей вершины v до цели:

$$h(v) \le h_{truth}(v) \quad (9)$$

, пропуская таким образом лучшие решения из-за их неоправданно завышенной оценки.

Монотонность же в свою очередь гарантирует, что для каждого ребра (u,v) графа G, эвристика $h(u)$ будет удовлетворять следующему условию:

$$h(u) \le c(u,v) + h(v) \quad (10)$$

где: $c(u,v)$ – равная весу ребра стоимость перехода из текущей ячейки u в соседнюю к ней проходимую ячейку v; $h(v)$ – эвристическая оценка от вершины v до цели (см. пункт 3).

Формула (10) означает, что переход из u в v и дальнейшее движение от неё должно быть дороже по стоимости, чем двигаться напрямую с точки зрения эвристики – это и есть монотонность, которая предотвращает уменьшение приоритета $f=g+h$ (см. пункт 3) при перемещении по графу и, как следствие, исключает повторное рассмотрение его предыдущих узлов.

Если же всё-таки попытаться учесть этот штраф $(1+\alpha * c(v))$ при расчёте веса w ребра, то скорректировать эвристику $h(v)$ будет довольно сложно, что может привести к потере оптимальности всего алгоритма A* в целом. Если же оставить вес ребра равным чистому расстоянию $d(u,v)$, не учитывая штраф, то эвристика будет правильной, но планировщик при этом может начать приближаться к потенциально опасным зонам;

3. Как и в случае очереди Q для Dijkstra, формируется приоритетная очередь O, которая содержит в себе значения так называемой функции приоритета $f(v)$ в порядке их возрастания:

$$f(v)=g(v)+h(v) \quad (11)$$

где: $g(v)$ в начале работы алгоритма формируется по тем же правилам, что и в Dijkstra (см. пункт 3 в предыдущем списке); $h(v)$ – эвристическая функция, которая позволяет оценить стоимость перехода от ячейки v до целевой точки пути g по следующим основным метрикам:

$$h(v)=\sqrt{(x_v-x_g)^2+(y_v-y_g)^2} \quad (12) - \textit{евклидова}$$

$$h(v)=|x_v-x_g|+|y_v-y_g| \quad (13) - \textit{манхэттенская}$$

$$h(v)=max(|x_v-x_g|,|y_v-y_g|) \quad (14) - \textit{диагональная}$$

4. Оставшиеся пункты 4-8 работы алгоритма A* будут отличаться от аналогичных им пунктов 4-8 работы алгоритма Dijkstra только тем, что работа будет вестись не с Q, а с O, и новая стоимость gn перехода из u в v будет рассчитываться не по (8), а по следующей формуле:

$$gn=g(u)+w(u,v) \quad (15)$$

где: $g(u)$ – накопленная стоимость перехода от старта s до текущей ячейки u с учётом всех предыдущих шагов; $w(u,v)=d(u,v)$ – стоимость перехода из текущей ячейки u в соседнюю к ней проходимую ячейку v **БЕЗ** учёта стоимости второй из Costmap (не путать со стоимостью ячейки и с накопленной стоимостью всего пути!).

Какой из двух описанных выше алгоритмов выбрать зависит от: 1) размерности сетки (Dijkstra обходит карту полностью), 2) её сложности (эвристика A* в виде евклидового расстояния может ошибочно проигнорировать нужный нам путь) и, как следствие, 3) производительности («звёздочка» быстрее).

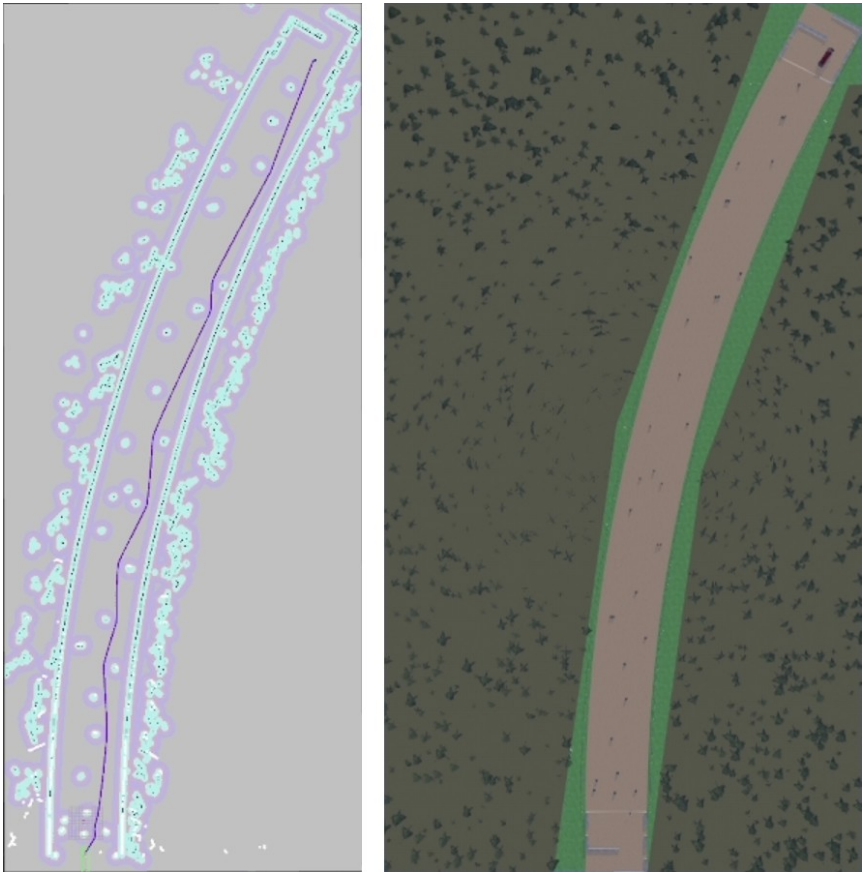

Рис. 15. Глобальный путь (изгибающаяся линия слева), построенный NFP

Regulated Pure Pursuit

Regulated Pure Pursuit (далее – RPP) [19] является «регулируемой» в плане линейных скоростей версией исходного локального планировщика *Pure Pursuit*, устроенного следующим образом:

1. На вход подаётся список точек пути P, представленный их двумерными координатами X, Y и полученный от NFP;
2. Определяется ближайшая к текущему месторасположению робота точка p_r на траектории (см. Рис. 16), а от неё так называемая точка упреждения (англ. *lookahead point*) p_l по следующим формулам:

$$dist(p_i) = \sqrt{(x_r - x_i)^2 + (y_r - y_i)^2} \quad (16)$$

$$p_l = p_i \in P, \begin{cases} dist(p_{i-1}) < L \\ dist(p_i) \geq L \end{cases} \quad (17)$$

где: L – дистанция упреждения (англ. *lookahead distance*) между двумя точками, задаваемая в параметрах алгоритма;

3. Преобразовав координаты точки упреждения в систему координат робота, вычисляем кривизну траектории k как:

$$k = \frac{2y_l'}{L^2} \quad (18)$$

где: y_l' – поперечная координата в системе координат робота;

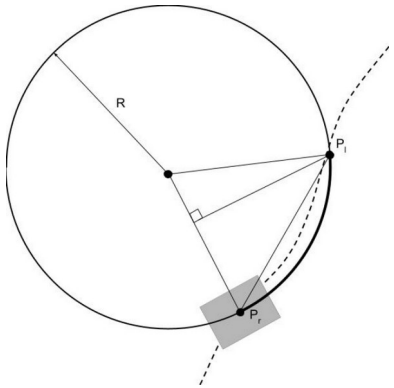

Рис. 16. Геометрия нахождения кривизны траектории

4. Рассчитывается управляющее воздействие в виде линейной скорости v_t как максимум из двух следующих эвристик:

$$v_t' = \left\{ \begin{array}{c} v_t \; при \; k > T_k, \\ \dfrac{v_t}{r_{min} k} \; при \; k \le T_k \end{array} \right\} \text{ (19) – эвр. кривизны (англ. } \textit{curvature}\text{)}$$

$$v_t' = \left\{ \begin{array}{c} v_t \dfrac{\alpha d_O}{d_{prox}} \; при \; d_O \le d_{prox}, \\ v_t \; при \; d_O > d_{prox} \end{array} \right\} \text{ (20) – эвр. сближения (англ. } \textit{proximity}\text{)}$$

где: r_{min} – минимально допустимый для робота радиус поворота; T_k – минимальный порог кривизны траектории, с которого начинает применяться первая эвристика; $\alpha \le 1.0$ – множитель усиления второй эвристической функции (чем он выше, тем быстрее робот снижает свою скорость вблизи препятствий); d_O – текущее расстояние до препятствия; d_{prox} – минимальное расстояние до препятствий, с которого начинает применяться вторая эвристика;

5. Рассчитывается управляющее воздействие в виде угловой скорости по следующей формуле:

$$\omega_t = v_t' k \text{ (21)}$$

Что оригинальный алгоритм, что его разновидности могут использоваться для следования локальному пути роботами со всенаправленным (англ. *omnidirectional*) движением, однако они будут ограничены в выполнении боковых перемещений; роботами с дифференциальным (англ. *differential*) приводом, которые могут следовать по любой *голономной* (свободной от кинематических ограничений) траектории, но что самое главное, RPP может использоваться для следования локальному пути роботами с *кинематикой Акерманна* (автомобильной кинематикой), что было подтверждено на практике (см. Рис. 17):

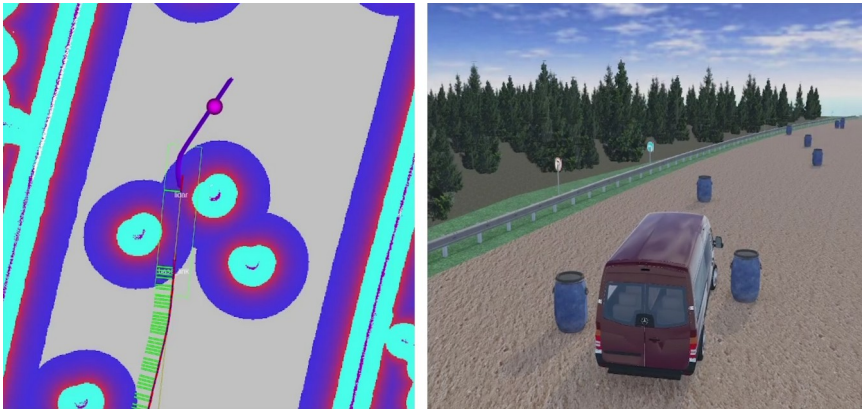

Рис. 17. RPP следует пути от NFP в виде линии с точкой (lookahead point) на конце

Model Predictive Path Integral

Model Predictive Path Integral (далее – MPPI) [20] является преемником алгоритма *TEB* [21], а также *MPC* [22], и так называемым *локальным контроллером*, работающим на основе предсказания и оптимизации движения в режиме реального времени, а также точного следования маршруту с учётом препятствий и кинематических ограничений.

Так же, как и для Regulated Pure Pursuit, глобальную траекторию для MPPI строит NavFn Planner, в то время как локальный планировщик работает в пределах заданного окна вокруг текущего положения робота следующим образом:

1. Генерируется набор управляющих воздействий (линейные и угловые скорости) путём добавления к ним случайного шума из *гауссовского распределения*. Каждая пара таких скоростей – это последовательность действий алгоритма длиной в T шагов вперёд (так называемый *горизонт планирования*, который задаётся параметрами *time_steps* и *model_dt*);
2. Эти действия прогоняются через динамическую модель робота с учётом его кинематики, на выходе выдавая множество траекторий движения;
3. Каждая из полученных траекторий оценивается по следующей формуле:

$$S_i = \sum_{t=0}^{T} c\left(x_t^{\,i}, u_t^{\,i}\right) \quad (22)$$

где: S_i – *стоимость* i-ой траектории; $x_t^{\,i}$ – состояние на шаге t для траектории i; $u_t^{\,i}$ – управляющее воздействие на этом шаге; c – один из множества так называемых *критиков* или функций стоимости (учёт кинематики и препятствий, следование цели, отклонения от пути и т. д.);

4. Лучшее управляющее воздействие из всех отбирается с использованием функции *softmax* следующим образом:

$$\delta u = \frac{\sum_{i=1}^{N} \exp\left(-\frac{1}{\lambda} S_i\right) * \epsilon_i}{\sum_{i=1}^{N} \exp\left(-\frac{1}{\lambda} S_i\right)} \quad (23)$$

где: δu – величина корректировки крайнего управляющего воздействия; ϵ_i – шум; λ – значение параметра *temperature*, регулирующего чувствительность алгоритма к стоимости (чем ближе к нулю, тем больше принимаются во внимание траектории с меньшей стоимостью);

5. Полученная корректировка используется на следующей итерации, влияя на уровень генерируемого шума и, как следствие, на очередной набор из линейных и угловых скоростей.

На практике (см. Рис. 18), готовая реализация Model Predictive Path Integral Controller из фреймворка Nav2, с грамотно подобранными параметрами, так же, как и RPP, неплохо показала себя, задев всего одно препятствие из 33-х на протяжении всего маршрута движения виртуального БНТС по испытательному полигону (см. Рис. 10).

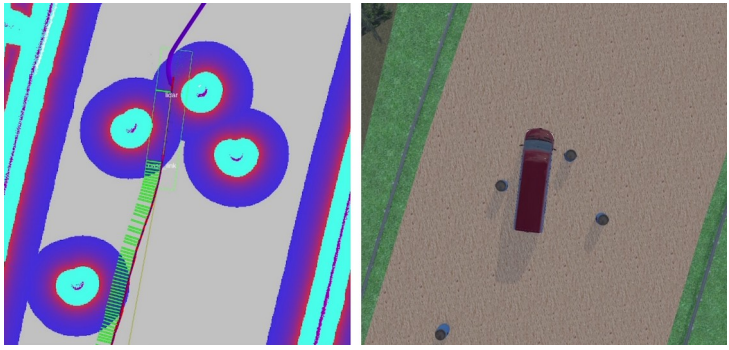

Рис. 18. MPPI следует пути от NFP на участке глобальной карты (слева)

Помимо роботов с кинематикой Акерманна, MPPI также поддерживает всенаправленных и дифференциальных роботов, что задаётся таким параметром алгоритма как *motion_model*.

Основной недостаток данного локального планировщика, который особенно сильно проявляется на более слабом железе, – это его высокая вычислительная сложность *O(N * T)* из-за массового сэмплирования десятков, а то и сотен траекторий движения и их оценивания, где: *N* – количество траекторий, T – временной горизонт планирования. Для сравнения, сложность RPP – *O(1)* из-за минимального объёма логики управления, отсутствия оптимизации и моделирования движения, а также простой структуры эвристик.

Заключение

В рамках данной работы был разработан метод, который позволяет на основе системы двухмерного кругового обзора строить сегментированную локальную карту и планировать по ней маршрут движения беспилотного наземного транспортного средства [23].

В процессе была подготовлена виртуальная среда симуляции с возможностью калибровки используемых в ней стереокамер; на основе данных с них была разработана система двухмерного кругового обзора, а затем построена сегментированная локальная карта с применением инструментов машинного и глубокого обучений.

Объединение предложенных готовых алгоритмов планирования маршрута движения БНТС с форматом построенной локальной карты – это та задача, которую только предстоит выполнить, хотя большая часть наработок по ней уже в том или ином виде присутствует в данном, а также в сторонних решениях собственной разработки.

Дальнейшие возможные исследования могут быть направлены на интеграцию предложенного подхода с трёхмерными системами кругового обзора и разработку гибридных моделей [24], сочетающих в себе различные источники данных для повышения качества картирования и навигации беспилотных наземных, а может быть и не только, транспортных средств.

Список литературы

1. D. J. Yeong, G. Velasco-Hernandez, J. Barry, J. Walsh, "Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review," Sensors. – 2021. – Vol. 21, No. 6. – Article 2140. – [Электронный ресурс]. – Режим доступа: https://www.mdpi.com/1424-8220/21/6/2140 (дата обращения: 12.05.2025);
2. Cyberbotics: Robotics simulation with Webots – [Электронный ресурс]. – Режим доступа: https://cyberbotics.com/ (дата обращения: 12.05.2025);
3. ROS: Home – [Электронный ресурс]. – Режим доступа: https://www.ros.org/ (дата обращения: 12.05.2025);
4. D. C. Brown, "Decentering distortion of lenses," Photogrammetric Engineering. – 1966. – Vol. 32, No. 3. – P. 444–462. – [Электронный ресурс]. – Режим доступа: https://web.archive.org/web/20180312205006/https://www.asprs.org/wp-content/uploads/pers/1966journal/may/1966_may_444-462.pdf (дата обращения: 13.05.2025);
5. Cambridge in Colour – Photography Tutorials & Learning Community, "Коррекция искажений, вносимых объективом" – [Электронный ресурс]. – Режим доступа: https://www.cambridgeincolour.com/ru/tutorials-ru/lens-corrections.htm (дата обращения: 13.05.2025);
6. ZED 2 - AI Stereo Camera – [Электронный ресурс]. – Режим доступа: https://www.stereolabs.com/en-fi/products/zed-2 (дата обращения: 14.05.2025);
7. Z. Zhengyou, "A Flexible New Technique for Camera Calibration" – [Электронный ресурс]. – Режим доступа: https://www.microsoft.com/en-us/research/publication/a-flexible-new-technique-for-camera-calibration/ (дата обращения: 13.05.2025);
8. OpenCV – Open Computer Vision Library – [Электронный ресурс]. – Режим доступа: https://opencv.org/ (дата обращения: 13.05.2025);
9. H. Yan, "Surround View System Introduction" – [Электронный ресурс]. – Режим доступа: https://github.com/hynpu/surround-view-system-introduction (дата обращения: 14.05.2025);
10. E. Zhang, "Fast Semantic Segmentation" – [Электронный ресурс]. – Режим доступа: https://github.com/ekzhang/fastseg (дата обращения: 14.05.2025);
11. T. Merkulova, B. Jayakumar, "Evaluation framework for Image Segmentation Algorithms," – 2025. – [Электронный ресурс]. – Режим доступа: https://arxiv.org/abs/2504.04435 (дата обращения: 14.05.2025);
12. K. Robison, "OpenAI cofounder Ilya Sutskever says the way AI is built is about to change," – 2024. – [Электронный ресурс]. – Режим доступа: https://www.theverge.com/2024/12/13/24320811/what-ilya-sutskever-sees-openai-model-data-training (дата обращения: 14.05.2025);
13. J. Redmon, S. Divvala, R. Girshick, A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," – 2015. – [Электронный ресурс]. – Режим доступа: https://arxiv.org/abs/1506.02640 (дата обращения: 14.05.2025);

14. CVAT: Leading Image & Video Data Annotation Platform – [Электронный ресурс]. – Режим доступа: https://www.cvat.ai/ (дата обращения: 14.05.2025);
15. Kaggle: Your Machine Learning and Data Science Community – [Электронный ресурс]. – Режим доступа: https://www.kaggle.com/ (дата обращения: 14.05.2025);
16. Macenski et al., "SLAM Toolbox: SLAM for the dynamic world," Journal of Open Source Software. – 2021. – 6(61). – 2783. – [Электронный ресурс]. – Режим доступа: https://doi.org/10.21105/joss.02783 (дата обращения: 15.05.2025);
17. Nav2 Navigation System – [Электронный ресурс]. – Режим доступа: https://nav2.org/ (дата обращения: 15.05.2025);
18. NavFn Planner – Nav2 1.0.0 documentation – [Электронный ресурс]. – Режим доступа: https://docs.nav2.org/configuration/packages/configuring-navfn.html (дата обращения: 24.05.2025);
19. S. Macenski, S. Singh, F. Martin, J. Gines, "Regulated Pure Pursuit for Robot Path Tracking," – 2023. – [Электронный ресурс]. – Режим доступа: https://arxiv.org/abs/2305.20026 (дата обращения: 15.05.2025);
20. G. Williams, P. Drews, B. Goldfain, J. M. Rehg, E. A. Theodorou, "Aggressive driving with model predictive path integral control," IEEE International Conference on Robotics and Automation (ICRA). – 2016. – [Электронный ресурс]. – Режим доступа: https://ieeexplore.ieee.org/document/7487277 (дата обращения: 15.05.2025);
21. teb_local_planner - ROS Wiki – [Электронный ресурс]. – Режим доступа: https://wiki.ros.org/teb_local_planner (дата обращения: 15.05.2025);
22. C. Rösmann, A. Makarow, T. Bertram, "Online Motion Planning based on Nonlinear Model Predictive Control with Non-Euclidean Rotation Groups," – 2020. – [Электронный ресурс]. – Режим доступа: https://arxiv.org/abs/2006.03534 (дата обращения: 15.05.2025);
23. A. Molchanov, "Surround View SegBEV" – [Электронный ресурс]. – Режим доступа: https://github.com/ghub-ayrtom/surround-view-segbev (дата обращения: 16.05.2025);
24. Z. Liu, H. Tang, A. Amini, X. Yang, H. Mao, D. Rus, S. Han, "BEVFusion: Multi-Task Multi-Sensor Fusion with Unified Bird's-Eye View Representation," – 2022. – [Электронный ресурс]. – Режим доступа: https://arxiv.org/abs/2205.13542 (дата обращения: 15.05.2025).