УДК 004.896

Е. В. Корягин, О. В. Толстель,
Д. Н. Хуторной, А. Г. Челядинский

МАТЕМАТИЧЕСКАЯ МОДЕЛЬ РОБОТА-АВТОМОБИЛЯ В СИМУЛЯТОРЕ GAZEBO

Представлены промежуточные результаты разработки системы управления робота-автомобиля в виртуальной среде Gazebo. Рассмотрен прототип физической модели робота. Изучены алгоритмы планирования и картографии.

Interim results of the development of the control system of the robot vehicle in a virtual environment Gazebo are presented. A prototype of a physical model of the robot is considered. Scheduling algorithms and mapping are studied.

Ключевые слова: роботизированный автомобиль, система принятия решений, микроконтроллер, блок управления, система навигации, картографирование.

Key words: robotic car, decision-making system, microcontroller, control unit, navigation system, mapping.

Для создания более совершенной математической модели робота-автомобиля была разработана его физическая модель. В качестве основы этого робота была взята модель радиоуправляемого автомобиля HSP, а именно весь ее «нижний уровень». «Верхний уровень» был полностью переработан и состоял из следующего набора элементов:

— модуль для определения положения автомобиля на трассе, состоящий из четырех аналоговых датчиков линии в передней части конструкции и двух датчиков, расположенных на заднем бампере;

— инфракрасный дальномер Sharp;

— Sensor Shield для Arduino Mega (плата-расширение для подключаемых компонентов);

— инфракрасный приемопередатчик данных, работающий по протоколу IrDA SIR (скорость передачи данных 115.2 Kbit/s;

— Arduino Mega 2560;

— драйвер моторов SyRen (одноканальный, 10A).

Итоговые габариты робота (рис. 1) не превышали начальных и составили 450 × 200 × 120 см (длина — ширина — высота).

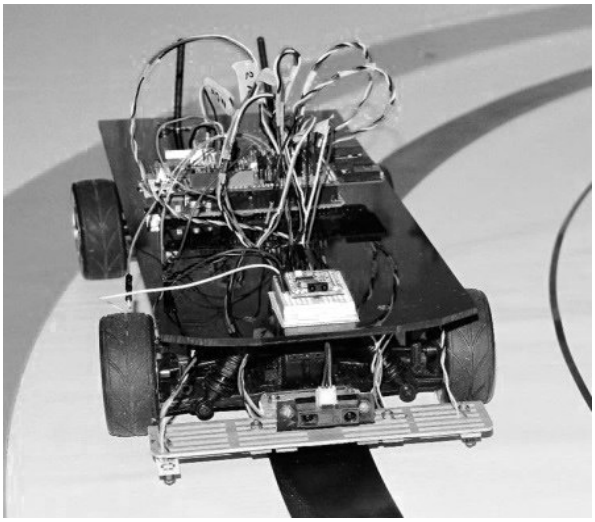

Рис. 1. Модель робота-автомобиля без корпуса

Центром всей системы выступила плата Arduino Mega2560. Вся информация с датчиков стекалась на плату, где происходили основные вычисления.

Обмен данными между платой Arduino и датчиками происходил по протоколу UART, в частности, для распознавания дорожных знаков и сигналов светофора была разработана следующая таблица кодов.

Коды для распознавания дорожных знаков и сигналов светофора

Передаваемые биты	Код	Расшифровка
0b00000001	0	Красный
0b00000010	1	Красный с желтым
0b00000011	2	Зеленый
0b00000100	3	Мигающий зеленый
0b00000101	4	Желтый
0b00001010	10	Пешеходный переход
0b00001011	11	Знак «Стоп»

В качестве логики движения (в том числе и системы принятия решений) использовалась система правил, в которой были расставлены следующие приоритеты (по убыванию важности): непредвиденное появление на дороге препятствия (пешеход) — знаки светофора (каждый также делился по приоритетам) — дорожные знаки.

В ходе испытания робота-автомобиля разработаны два вида полей: для тестирования маневренности и скоростных показателей автомобиля и для отладки поведения робота в условиях насыщенного городского движения.

Основным недостатком такой модели была ее привязанность к заданной трассе, а именно наличие черной полосы. Поэтому для разрешения этой проблемы была сделана математическая модель.

Навигация в ROS (в частности Gazebo) разделяется на три отдельных модуля:

1) глобальный планировщик маршрута;

2) локальный планировщик маршрута;

3) модуль выхода из тупиковых ситуаций.

Глобальный планировщик строит общий план маршрута, определив на нем путевые точки. Далее этот план передается локальному планировщику. Последний отвечает за оптимальное перемещение робота по точкам и содержит алгоритмы для прокладывания наилучшего пути между препятствиями.

Используется алгоритм динамического окна (DWA — Dynamic Window Approach) (рис. 2):

— выполняется моделирование движения робота по каждой из выбранных скоростей, если применять эти данные в небольшой промежуток времени, можно предсказать поведение робота;

— по результатам моделирования анализируются все возможные траектории с использованием таких критериев, как близость к препятствиям, к цели, глобальный путь и скорость; в итоге маршруты, ведущие к столкновениям, отбрасываются;

— выбирается траектория с лучшими показателями, и значения скоростей передаются роботу.

Алгоритм повторяется, пока робот не достигнет цели.

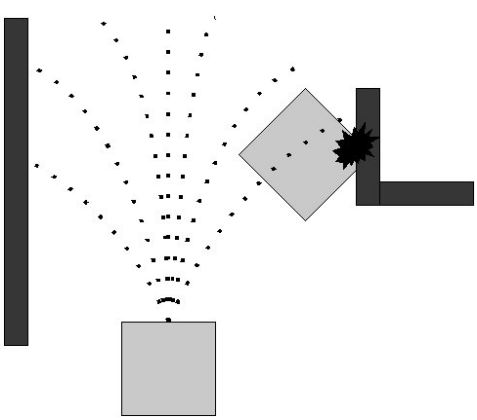

Рис. 2. Выбор траекторий алгоритма DWA

Данный алгоритм не указывает скорость в конечных точках маршрута, но допускает настройку таких показателей, как максимальные линейная и угловая скорость, ускорение и стоимость прохождения пути, что позволяет видоизменять генерируемые траектории.

Навигация ROS предполагает движение «от точки к точке», при этом робот разгоняется (не выше $V_{\max}$) и тормозит с ускорением a. Поскольку скорости в конечных точках маршрута не задаются, это можно сделать косвенно. Например, роботу достаточно передать координаты O^*, лежащей дальше от последней точки (рис. 3), чтобы иметь в точке O заданные скорость и направление. Осталось вычислить эти координаты.

Рассчитываются они следующим образом:

$$x^* = x + l\cos\alpha,$$

$$y^* = y + l\sin\alpha.$$

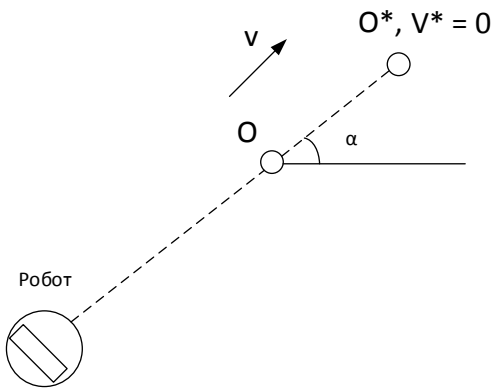

Рис. 3. Фиктивная точка пути для обеспечения скорости в заданной точке

Учитывая, что

$$l = \frac{V_x^2 + V_y^2}{2a_m},\ \sin\alpha = \frac{V_y}{\sqrt{V_x^2 + V_y^2}},\ \cos\alpha = \frac{V_x}{\sqrt{V_x^2 + V_y^2}},$$

получим окончательные формулы

$$x^* = x + \frac{V_x^2 + V_y^2}{2a_m}\frac{V_x}{\sqrt{V_x^2 + V_y^2}} = x + \frac{V_x\sqrt{V_x^2 + V_y^2}}{2a_m} = x + \frac{V_x V}{2a_m},$$

$$y^* = y + \frac{V_y V}{2a_m}.$$

При достижении роботом заданной точки команда «двигаться в фиктивную точку» отменится, и робот продолжит путь к следующей точке маршрута.

В данной работе используется стандартный модуль картографирования Gmapping, работающий по технологии SLAM (Simultaneous Localization And Mapping, одновременная локализация и картографирование). Эта технология делает возможным одновременное построение карты и определение местоположения на ней робота, имеющего устройство лазерного сканирования. Пример такой карты показан на рисунке 4.

Рис. 4. Пример работы модуля картографии

Модуль картографирования может работать как с симуляцией, так и с реальным роботом.

Для разработки виртуальной модели и окружения использовались инструменты симулятора Gazebo. Для создания упрощенной модели а/м (без текстур) — открытое программное обеспечение для трехмерного моделирования Blender. Blender позволяет экспортировать созданные модели в xml-подобный формат sdf, который поддерживается Gazebo. Для тестирования алгоритмов был создан виртуальный мир, представляющий собой перекресток двух дорог, в который так же были добавлены объекты — промышенные здания, дорожные знаки, препятствия (рис. 5—7).

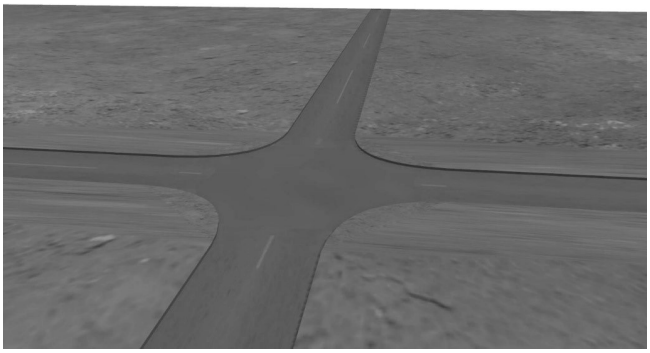

Рис. 5. Основа созданного виртуального мира

Рис. 6. Объекты инфраструктуры в модели

Рис. 7. Общий план созданного виртуального мира

В дальнейшем планируется усложнить условия тестирования — добавить больше препятствий в виртуальный мир.

Для работы модели в инфраструктуре robot operating system были созданы два вспомогательных пакета ROS — myrobo_gazebo и myrobo _description.

Для описания используется формат описания робототехнических систем URDF (Universal Robot Description Format), который базируется на xml и содержит стандартизованную характеристику параметров автомобиля — веса, габаритов шасси и колес, работы амортизаторов.

Пакет myrobo_gazebo имеет необходимые интеграционно-конфигурационные файлы для взаимодействия и совместной работы программного обеспечения инфраструктуры ROS, управляющего программного обеспечения и симулятора Gazebo. Помимо конфигурационных файлов пакет содержит описания виртуальных сред для симуляции и скрипт-дополнение для Gazebo на языке Python, через который осущест-

вляется управление моделью автомобиля внутри симулируемого мира. Пакеты специально скомпонованы для поддержки и быстрой разработки множества моделей автомобилей и виртуальных миров.

Для рассматриваемой модели были проведены интеграционные тесты на взаимодействие с программными компонентами ROS и с разрабатываемым алгоритмом. Модель успешно взаимодействует с набором навигационных пакетов по схеме, представленной на рисунке 8.

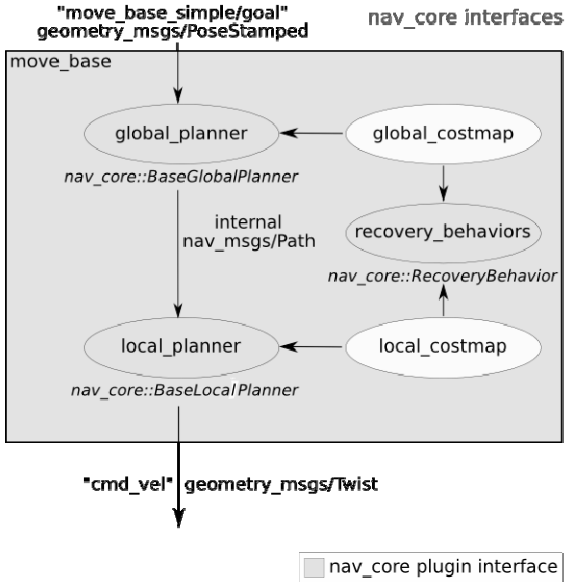

Рис. 8. Диаграмма взаимодействия пакетов навигации на основе ROS

Вместе с использованием стандартных пакетов среды ROS применяются алгоритмы обнаружения препятствий собственной разработки, а также программная модель управления.

В настоящий момент успешно интегрирован вспомогательный узел ROS dwa_local_planner, который позволяет избегать препятствий в ближайшем окружении автомобиля и планировать движение в локальном пространстве. Габариты и прочие необходимые для планирования движения параметры автомобиля программный код узла извлекает из конфигурационных файлов модели.

Был составлен набор изображений (датасет), включающий в себя выборочные изображения из следущих датасетов:

1) ETH pedestrain dataset;

2) Penn-Fudan Database for pedestrian detection;

3) INRIA pedestrian dataset;

4) изображения, самостоятельно отснятые с камеры в городе.

Затем общий датасет был разбит на две части в соотношении 2 : 1; большая часть использовалась для обучения классификаторов, меньшая для тестирования классификаторов.

В качестве алгоритма, позволяющего оптимально решать поставленную задачу, рассматривались:

1) метод Виолы — Джонса (Haar + Adaboost);

2) HOG + SVM;

3) HOG + AdaBoost;

Метод Виолы — Джонса применялся для поиска пешеходов. Для обучения классификации используются признаки Хаара, описывающие разницы суммированных интенсивностей в прямоугольных областях, вычисляемые с помощью представления изображения в интегральном виде (признаки очень быстро рассчитываются за константное время).

Алгоритм адаптивного бустинга Adaboost позволяет строить композицию классификаторов, когда результат каждого предыдущего улучшает результат последующего, при этом акцент делается на объекты, которые распознаются хуже (также это называется слабой моделью). Ситуацию применения алгоритма к задаче можно описать следующим образом. Метод обладает высокой скоростью и высокой точностью и позволяет собственно разделять два класса: объекты искомого класса и объекты, не принадлежащие искомому классу. Движение человека у проезжей части отличается достаточно большим разнообразием поз тела, поэтому возникает вопрос, как обучить классификатор по методу Виолы — Джонса исходя из этого (такое допущение о необходимом разнообразии поз делается из-за специфики признаков Хаара, которые при описании не смогут наглядно описывать объекты разной формы как объекты одного класса).

Существует два варианта ответа на этот вопрос.

1. Группировать обучающую выборку изображений по типовым позам движения пешехода и обучать классификатор под каждую из поз (как это сделано в библиотеке OpenCV для разных частей лица и туловища). Затем при обработке каждого кадра последовательно обрабатывать изображение несколькими классификаторами, обученными под типовую позу движения пешехода. Главным минусом будет медленная скорость работы, так как несколько классификаторов будут последовательно работать друг за другом (по той вышеуказанной причине, что метод позволяет разделять только два класса: объекты искомого класса и другие объекты неискомого класса).

2. Брать всю выборку изображений и по ней обучать один классификатор для поиска всех вариаций движения пешехода. Для обработки одного кадра будет достаточно работы одного классификатора. Несмотря на высокую скорость работы минусом такого решения станет то, что классификатор часто будет ложно срабатывать на объекты, не являющиеся пешеходами (ввиду большой вариации поз пешеходов при движении).

После непродолжительного тестирования даже на глаз (без построения оценивающей метрики) стало ясно, что метод дает очень большое количество ложных срабатываний.

Вывод. Метод Виолы — Джонса был бы идеален для стабильных объектов с очень малой вариацией форм. В целом признаки Хаара не-

достаточно информативны для того, чтобы описать движение пешехода, и в результате обучения классификатор «переобучивается» и дает большое количество ложных срабатываний на тестовой выборке.

Необходимо экспериментировать с подбором другого алгоритма или изменением существующего. В качестве признаков, описывающих объекты, можно заменить признаки Хаара на HOG-дескрипторы, использующие гистограммы направленных градиентов (Histogramms of oriented gradients, HOG), а в качестве классификатора выбрать SVM (метод опорных векторов) вместо каскадного классификатора.

Основной идеей алгоритма HOG-дескриптора является допущение, что внешний вид и форма объекта на участке изображения могут быть описаны распределением градиентов интенсивности или направлением краев (рис. 9). Реализация этих дескрипторов может быть произведена путем разделения изображения на маленькие связные области, именуемые ячейками, и расчетом для каждой ячейки гистограммы направлений градиентов или направлений краев для пикселов, находящихся внутри ячейки. Комбинация этих гистограмм и есть дескриптор. Для увеличения точности локальные гистограммы подвергаются нормализации по контрасту. С этой целью вычисляется мера интенсивности на большем фрагменте изображения, который называется блоком, и полученное значение используется для нормализации. Нормализованные дескрипторы обладают лучшей инвариантностью по отношению к освещению.

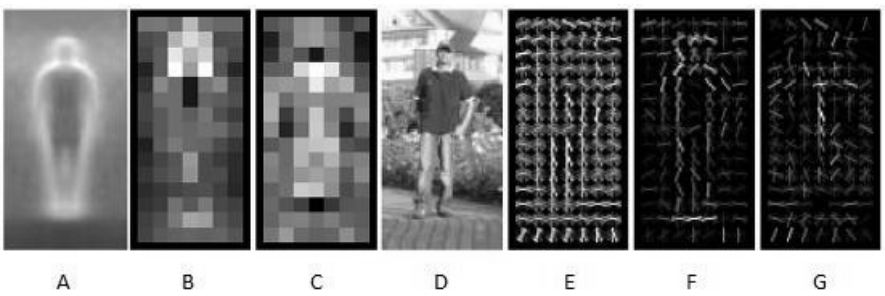

Рис. 9. Описание HOG-дескриптора:
A — усредненный контур; B — цвет клетки равен максимальному положительному весу для пикселей клетки; C — цвет клетки равен максимальному модулю из отрицательных весов для пикселей клетки; D — исходное изображение; E — дескриптор изображения; F — дескриптор, домноженный на положительные веса; G — дескриптор, домноженный на отрицательные веса

Дескриптор HOG имеет несколько преимуществ по сравнению с другими дескрипторами. Поскольку HOG работает локально, метод поддерживает инвариантность геометрических и фотометрических преобразований, за исключением ориентации объекта. Подобные изменения появятся только в больших фрагментах изображения. Более того, как обнаружили авторы дескриптора Далал и Триггс, грубое разбиение пространства, точное вычисление направлений и сильная ло-

кальная фотометрическая нормализация позволяют игнорировать движения пешеходов, если они поддерживают вертикальное положение тела. Дескриптор HOG, таким образом, является хорошим средством нахождения людей на изображениях.

Метод опорных векторов (SVM, Support Vector Machine) — это линейный классификатор, очень часто и успешно применяющийся в машинном обучении. Задача обучения и классификации заключается в том, чтобы классификатор по вычисляемым признакам (HOG-дескрипторам) научился отличать пешеходов от непешеходов (в задаче классификации всего два класса). Основным принципами SVM являются:

1) максимизация межклассового расстояния. Разделяющая классы полоса выбирается не случайным образом, а единственным, так, чтобы ширина полосы была максимально возможной (рис. 10). Алгоритм предполагает, что чем шире разделяющая полоса, тем правильнее классифицируются объекты;

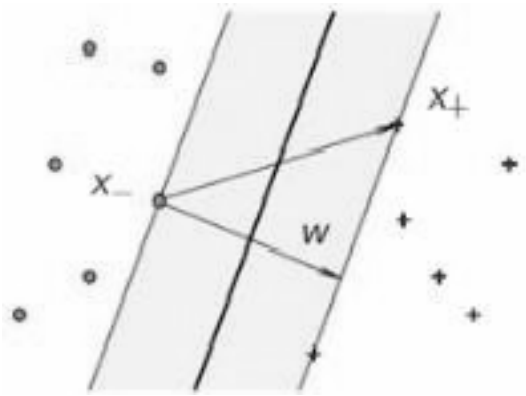

Рис. 10. Максимизация межклассового расстояния

2) в SVM (методе опорных векторов) векторы называются опорными, так как только они участвуют в построении классификатора. Объекты, векторы признаков которых не являются опорными, не участвуют в построении классификатора. Из набора условий теоремы Каруша — Куна — Таккера применимо к задаче SVM выводится три типа возможных объектов:

а) периферийные (неинформативные объекты, не участвующие в построении классификатора);

б) опорные граничные объекты;

в) опорные объекты — «нарушители» (этим и объясняется чувствительность SVM к шуму, и шумовые объекты участвуют в построении классификатора);

3) применение различных ядер (чаще всего типовых и уже реализованных в пакетах для машинного обучения), обладающих свойствами скалярного произведения. Важным свойством является то, что ядро может быть определено в пространстве, превышающем размерность исходного пространства признаков.

Преимущества SVM:

— один из наиболее быстрых методов нахождения решающих функций;

— сводится к решению задачи квадратичного программирования в выпуклой области, которое всегда является единственным;

— находит разделяющую полосу максимальной ширины, что позволяет в дальнейшем осуществлять более уверенную классификацию.

Недостатки SVM:

— метод чувствителен к шумам и стандартизации данных;

— не существует общего подхода к автоматическому выбору ядра (и построению спрямляющего подпространства в целом) в случае линейной неразделимости классов (оптимальное ядро выбрать можно по скользящему контролю, иначе именуемому cross-validation).

Преимущества и недостатки приведены для классического SVM (без учета так или иначе избавляющих от недостатков модификаций: RVM, SFM, RFM).

В качестве признаков используются уже описанные HOG-дескрипторы, а в качестве классификатора — AdaBoost. Метод не так часто применяется на практике, как HOG + SVM. Стоит отметить, что AdaBoost при построении алгоритмов классификации значительно дольше обучается, чем SVM. Кроме того, два AdaBoost-классификатора иногда находят пешеходов, которые не находит SVM, но также обнаруживают много объектов, не являющихся в действительности пешеходами.

Надежнее всего работает SVM-классификатор, обученный на HOG-дескрипторах, при его работе возникает наименьшее количество ложных срабатываний.

На рисунке 11 показаны примеры работы алгоритмов.

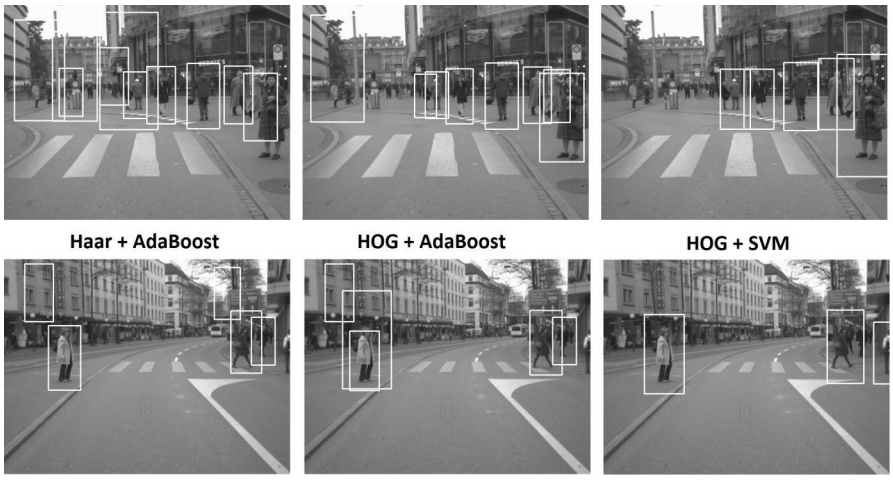

Рис. 11. Примеры работы алгоритмов

Существует помимо вышерассмотренных еще ряд алгоритмов, использующих HOG-дескриптор как признак (и его модификации, а также отличные от HOG признаки), а SVM или AdaBoost как классифика-

тор. Некоторые из них обладают более высокими качеством и скоростью работы и иногда ограниченной лицензией на право использования. В дальнейшем планируется углубленное изучение, тестирование, модификация и возможность применения подобных алгоритмов.

Программный комплекс решает задачи обнаружения пешеходов и автомобилей. Он состоит из двух основных модулей: детектора автомобилей и детектора пешеходов, а также взаимодействующих с ними компонентов ROS для работы с камерой, обработки и визуализации результатов работы.

Детектор пешеходов полностью реализован на языке C++, для обработки изображений использовалась библиотека OpenCV (рис. 12). Детектор автомобилей и остальные компоненты системы (за исключением детектора пешеходов) реализованы на языке Python с применением библиотек OpenCV, NumPy (рис. 12).

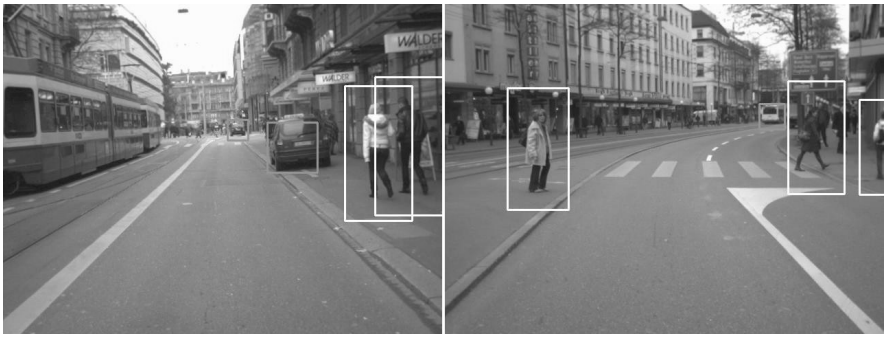

Рис. 12. Визуализация работы детекторов пешеходов и автомобилей

В терминологии средств разработки (ROS) программное средство называется пакетом (ROS package). Функционал системы выполняют узлы (ROS nodes), которые по топикам (ROS topics) обмениваются друг с другом входными и выходными данными или сообщениями (ROS messages). Процесс отправки выходных данных или сообщения(ий) называется публикацией (ROS publishing), а процесс приема входных данных или сообщения(ий) называется подпиской (ROS subscribing). Для большей гибкости и функциональности при обмене сообщений между узлами по топикам используются параметры (ROS parameters). Обмен сообщениями для одного конкретного узла по одному топику имеет односторонний характер: либо публикация, либо подписка. Также существуют сервисы (ROS services), которые очень похожи на сообщения, но позволяют обмениваться данными в двух направлениях, посылая запрос (request) и получая на него ответ (response).

В составе любого ROS-пакета можно выделить несколько главных моментов:

1) файлом package.xml описываются название пакета, зависимости сборки проекта и зависимости выполнения проекта, а также служебная информация: автор, тип лицензии ПО и характеристика пакета;

2) файл CmakeLists.txt служит для сборки проекта, в нем указывается название проекта, включаются директории поиска заголовочных файлов, осуществляется поиск необходимых зависимостей, определяется исполняемый файл и связываются необходимые библиотеки;

3) catkin — система сборки ROS-пакетов, которая очень похожа на CMake и по сути своей представляет CMake, отличаясь лишь парой системных переменных при сборке проектов и иерархией папок и файлов, получаемых в результате сборки проекта;

4) в папках src и include расположены соответственно файлы реализации *.cpp и заголовочные файлы *.h, при сборке этих файлов получаются исполняемые файлы и библиотеки на языке C++;

5) в папке scripts расположены скриптовые файлы на языке Python с расширениями *.py, *.pyc;

6) в папке launch расположены launch-файлы, которые описывают конфигурацию узлов запускаемого пакета, а также все параметры, использующиеся при запуске узлами для обмена сообщениями по топикам и другим целям. Чаще всего запуск пакетов и их работа осуществляется запуском launch-файлов.

Формат команды: **$ roslaunch [имя_пакета] [имя_launch_файла]**

Пример: **$ roslaunch people_and_car_detector myrobo_vision.launch**

Альтернативный способ запуска ROS-пакетов, а точнее, отдельных его узлов — использование команды rosrun.

Формат команды: **$ rosrun [имя_пакета] [имя_узла] <параметры>**

Пример: **$ rosrun image_transport_tutorial my_publisher /home/z/Desktop/image.jpg**

Конкретно в контексте разработанного программного комплекса это выглядит следующим образом:

1) ROS nodes (узлы): usb_cam, people_detector, people_detector_view, car_detector, car_detector_view, vision_control, rosout (на схеме нет, узел запускается при запуске обязательного процесса roscore);

2) ROS topics (топики): /usb_cam/image_raw, /image, /image2, /rois, /rosout (топик служебный от процесса roscore), /rosout_agg (служебный от roscore).

В начале работы системы от узла usb_cam (позволяющего работать с usb-камерами) по топику /usb_cam/image_raw передаются изображения с usb-камеры на узлы: people_detector и car_detector. Затем эти узлы, получив кадр изображения через топик /usb_cam/image_raw, независимо друг от друга обрабатывают его (работают асинхронно, так как детекторы используют разные алгоритмы, работают с разной скоростью, реализованы на разных языках: C++ и Python). После обработки кадра изображения каждый из двух детекторов публикует по два сообщения: обработанный кадр с обнаруженными объектами (по топикам /image и /image2) в узел визуализации (people_detector_view и car_detector_view) и координаты размеченных областей обнаруженных объектов в узел управления системой (vision_control).

Структура системы может незначительно меняться в зависимости от источника входящих в систему изображений, в роли источника могут выступать: 1) usb-камера; 2) ethernet-камера; 3) видеофайл.

Для того чтобы этот альтернативный выбор был возможен, создано несколько различных конфигураций узлов системы с помощью различных *.launch-файлов. Пользователь сам выбирает, какую конфигурацию он хочет запустить в зависимости от наличия usb-камеры, ethernet-камеры или видеофайла.

Для взаимодействия с алгоритмами навигации модель автомобиля дополнена симулируемыми сенсорами, которые воспроизводят поведение сенсоров в виртуальной среде. Симулируемые сенсоры представляют собой программные компоненты, которые работают в соответствии с описанием модели. В описание виртуальной модели автомобиля включены макросы камеры, лазерного дальномера и акселерометра. С помощью этих макросов и дополнений к среде Gazebo в модель добавлены симулируемая камера, лазерные дальномеры и акселерометр согласно рисунку 13. При необходимости с помощью макросов могут быть добавлены дополнительные виртуальные датчики. Свойства симулируемых датчиков также могут быть изменены с помощью редактирования конфигурации макроса, также могут быть изменены сами датчики.

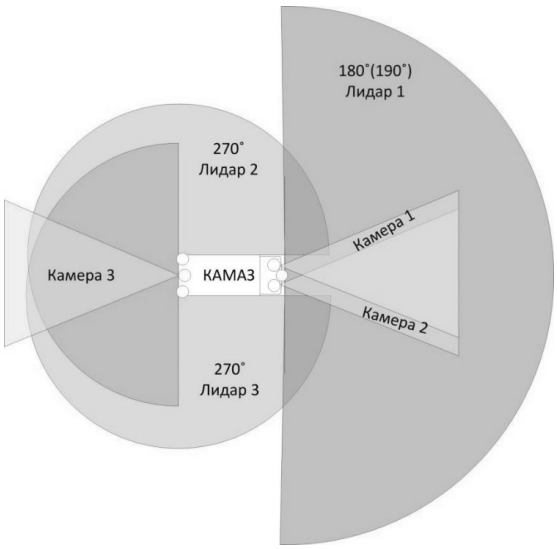

Рис. 13. Схема расположения датчиков на модели автомобиля

Данные с симулированных датчиков передаются через стандартные очереди (topic) системы ROS и считываются системами навигации и определения разметки.

Подсистема логики навигации основана на интеграции стандартных подсистем ROS и подсистемы обнаружения разметки. Для тестирования работы всего комплекса дополнительно стыкуется ранее разработанная система виртуальной модели автомобиля.

Для обработки логики навигации использован стандартный стек узлов Robot Operation System с необходимыми для функционирования модификациями.

Создан пакет ROS myrobo_nav, который управляет стандартным пакетом dwa_local_planner, передает ему задачи для навигации и реализует построение локальной карты cost_map, в результате чего локальный планировщик выдает вектор оптимального движения. Для планировщика создана конфигурация, отражающая особенности передвижения автомобилей, в которой определены ограничения на параметры вращения, ускорения, скоростей модели.

Список литературы

1. *Арзуметов А.М., Корягин Е.В., Орешков С.С., Толстель О.В.* Некоторые результаты моделирования управления автомобилем-роботом // Вестник Балтийского федерального университета им. И. Канта. 2012. Вып. 10. С. 94—98.

2. *Корягин Е.В.* Разработка системы управления мобильного робота на основе нейроподобной растущей сети // Там же. 2010. Вып. 10. С. 254—258.

3. *Корягин Е.В.* Разработка системы управления мобильного робота // Интеллектуальные системы AIS'10 : междунар. науч.-техн. конф. 2010. С. 90—94.

Об авторах

Евгений Викторович Корягин — ассист., Балтийский федеральный университет им. И. Канта, Калининград.
E-mail: koryagin.evgeniy@gmail.com

Олег Владимирович Толстель — канд. техн. наук, доц., Балтийский федеральный университет им. И. Канта, Калининград.
E-mail: oleg77764@mail.ru

Дмитрий Николаевич Хуторной — асп., Балтийский федеральный университет им. И. Канта, Калининград.
E-mail: dim99on@gmail.com

Александр Геннадьевич Челядинский — асп., Балтийский федеральный университет им. И. Канта, Калининград.
E-mail: chelyadinskiy@gmail.com

About the authors

Evgeniy Koryagin, ass., I. Kant Federal University, Kaliningrad.
E-mail: koryagin.evgeniy@gmail.com

Dr Oleg Tolstel', Ph.D., ass. prof., I. Kant Federal University, Kaliningrad.
E-mail: oleg77764@mail.ru

Dmitriy Hutornoy, PhD student, Kant Federal University, Kaliningrad.
E-mail: dim99on@gmail.com

Alexander Chelyadinskiy, PhD student, Kant Federal University, Kaliningrad.
E-mail: chelyadinskiy@gmail.com