1. **Этапы работы над проектом**
   1. **Подготовка к работе**

Языком программирования был выбрал “Python” (версии 3.8), т.к. работать с симулятором “gym-DuckieTown” удобнее именно на нём. Также он был выбрал по личным предпочтениям участников проекта, поскольку код на нём пишется быстрее, чем на том же “C++”.

Редактором кода была выбрана программа “Pycharm”, ибо она является довольно мощным инструментом для разработки, да к тому же её “community”-версия бесплатная.

* 1. **Этапы работы**

Сначала нужно было определиться с тем, какие примерно классы можно будет использовать. Первоначальный список классов был такой: “Node”, “RoadStretch”, “Graph”, “Driver” (однако позже появились новые классы, реализация которых теперь гораздо массивнее и сложнее, чем реализация данных классов).

Далее нужно было понять, как будет составляться граф. Первоначально планировалось, что в графе будут лишь узлы, которые показывают поворот (то есть если граф состоит только из прямого участка дороги, то будет в графе лишь 2 точки: начало участка дороги и его конец), однако такой способ составления графа негативно влияет на точность составления карты, по сколько тогда нельзя будет увидеть форму участка дороги, где бот поворачивает (ну или по крайней мере понимание этого для нас было не так просто, как понимание следующих методов). Однако грамотная реализация класса, с помощью которого бот сам ездит по центру дороги, не съезжая с него, улучшила бы точность.

Однако при работе с этим классом вдруг пришло понимание того, что если мы можем получить расстояние до ближайшей точки (с направлением, поскольку оно может быть как больше нуля, так и меньше), угол между ботом и дорогой (он также может быть больше и меньше нуля), то значит теоретически можно узнать, до какой точки (от бота) считается это ближайшее расстояние (так и было сделано в итоге). Был составлен алгоритм нахождения ближайшей точки дороги, которая гарантировано и однозначно (если бот движется) определяется текущим абсолютным положением бота, абсолютным углом бота, расстоянием до дороги (с учётом знака расстояния), углом между дорогой и ботом. Также для однозначного определения точки необходимо прошлое положение, но в расчётах оно не участвует, а лишь помогает выбрать из двух точек одну.

Для удобства был добавлен класс “Point” (для удобства работы с координатами). Далее был добавлен класс “CalculatorInputData”, который просто в одном месте хранит все данные, необходимые для нахождения ближайшей точки (текущее и прошлое положения бота, абсолютный угол бота, угол между ботом и дорогой, расстояние до дороги). Были созданы классы “Line”, “Vector” и “ClosestPointsCalculator”.

Далее были созданы классы “GraphDrawerSettings” и “GraphDrawer”. Был добавлен класс “Color” (для удобного хранения трёх цветов); класс “Driver” был разделён на “ManualDriver” (ручное управление ботом) и “AutoDriver” (автоматическая езда по дороге)

* 1. **Использованные технологии и ресурсы**

В работе над проектом помогли такие сайты, как:

* “Wolfram | Alpha” (<https://www.wolframalpha.com/>) – помогал упрощать выражения, решать уравнения
* “Matematikam” (<http://matematikam.ru/calculate-online/grafik.php>) – помогал строить графики функций, множества
* “OnlineMSchool” (<https://ru.onlinemschool.com/math/library/analytic_geometry/lines_angle/>) – отсюда была получена информация о нахождении угла между двумя прямыми

1. **Анализ полученного проекта**
   1. **Сопоставление целей и полученного продукта**

Проект выполняет поставленные задачи, а именно автоматически ездит и составляет карту дорог.

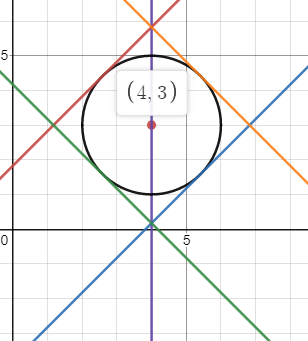
* 1. **Основные принципы работы алгоритма**

Рассмотрим вкратце все классы, используемые в нашем проекте:

* Point (точка) – хранение координаты точки
* Color (цвет) – удобство работы с цветами
* Vector (вектор) – работа с векторами
* Line (прямая) – работа с прямыми (который находятся в плоскости XY)
* GraphDrawerSettings (настройки рисования графа) – класс хранит все возможные параметры того, как будет рисоваться граф (цвета узлов, радиус точек, цвета линий и т.д.)
* CalculatorInputData (входные данные для калькулятора ближайшей точки) – хранит данные, необходимые для нахождения ближайшей точки
* Node (узел) – хранит лишь поле типа “Point”, нужен для удобства, ведь в графе находятся узлы, а не точки, также были мысли добавления других полей и методов в класс
* RoadStretch (участок дороги) – хранит 2 точки: начало и конец участка дороги
* Graph (граф) – хранит в себе узлы, участки, есть методы добавления новых узлов и т.д.
* GraphDrawer (чертёжник графа) – отвечает за рисования графа (и дальнейший показ получившейся карты)
* Driver (водитель) – родительский класс, отвечающий за движение, у него есть 2 дочерних класса: AutoDriver (авто-водитель) – класс, отвечающий за автоматическую езду по дороге; ManualDriver (ручной водитель) – класс, отвечающий за ручную езду по линии (нужен был для тестов)
* ClosestPointsCalculator (калькулятор ближайших точек) – класс находит ближайшую точку дороги, принимая на вход поле типа “CalculatorInputData”

Рассмотрим самое интересное: нахождение ближайшей точки дороги.

Пусть координаты бота сейчас = (4, 3), абсолютный угол = 90°, угол между ботом и дорогой = -45°, расстояние между ботом и дорогой = -2. Предыдущее положение было (1;1).

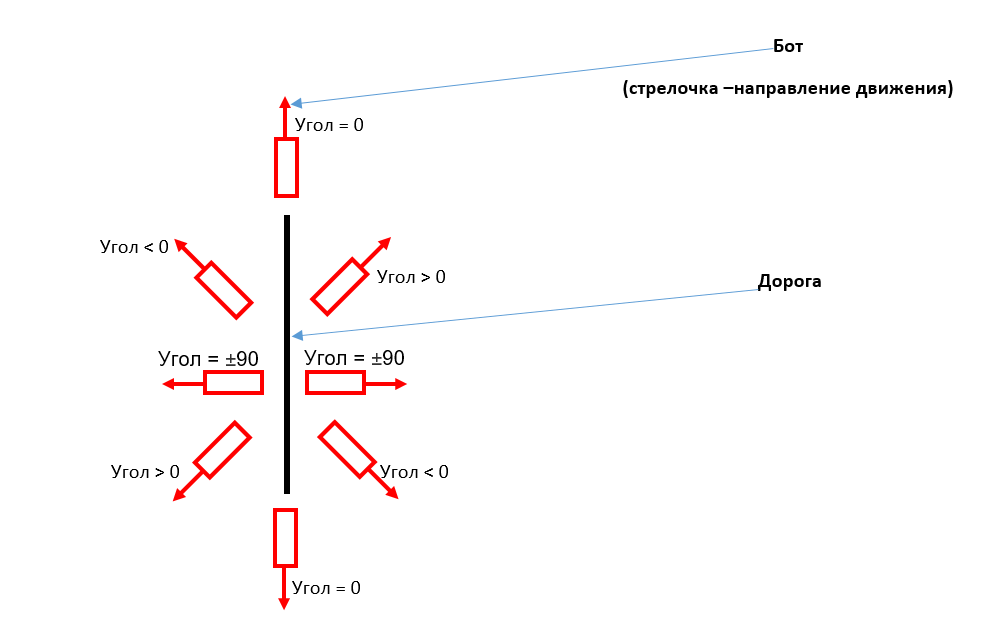
Построим геометрическую модель нашей ситуации на плоскости XY:

*Фиолетовая* линия (перпендикулярная OX и проходящая через точку (4;3)) – это линия, обозначающая нашего бота. Угол её наклона = 90° (то есть она перпендикулярна OX) и проходит через точку (4;3), то есть через текущую точку. Предыдущее положение было

*Черная окружность* – множество точек, которые находятся на расстоянии 2 от текущей точки, то есть это множество точек, среди которых есть та “ближайшая точка дороги”, которая нам нужна

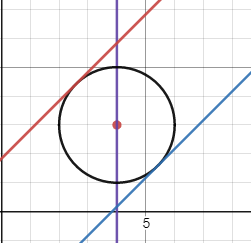
*Остальные прямые* – прямые, обозначающие дорогу

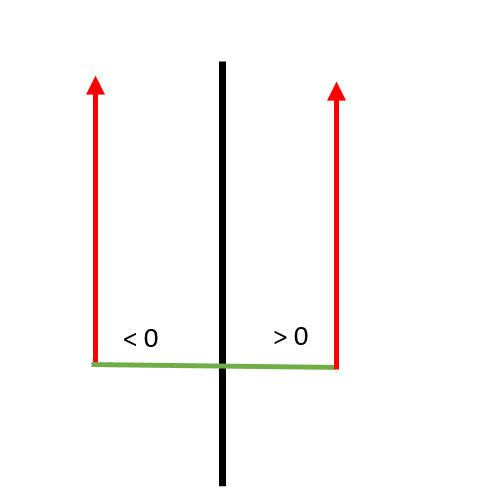
На нашем рисунке этих прямых 4, то есть точек, которые могут быть ближайшими (то лежащими на одной из этих прямых и на окружности) тоже 4, однако есть кое-что, что может их количество уменьшить до 1 (то есть до однозначно определяемой)

Во-первых, gym-DuckieTown возвращает угол между дорогой и ботом не всегда положительный, диапазон этих значений = [-90°; 90°], и вот определяется знак угла:

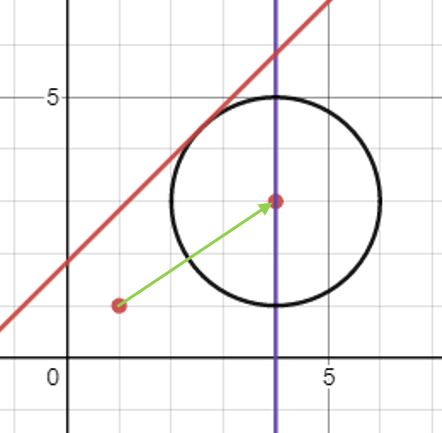
Благодаря этому можно отбросить 2 прямые (а значит и 2 возможные ближайшие точки), которые, судя по этой схеме, над не подходят.

После этого этапа остаётся 2 прямые, которые могут обозначать дорогу



Следующий этап – однозначное определение прямой, содержащей дорогу (а значит и ближайшей точки дороги). Дело в том, что значение расстояние до дороги тоже может вернуться как со знаком плюс, так и со знаком минус.

То есть если вектор движения бота справа от дороги, то расстояние будет положительным, если слева – отрицательным. Это и позволяет нам однозначно определить прямую, содержащую дорогу, а значит и ближайшую точку дороги.



Поскольку у нас осталась только 1 прямая, точка которой лежит на окружности нужного нам радиуса, то мы можем найти эту точку, и она и будет ближайшей точкой дороги.

То есть, для нас особо неважно, как бот ездит (по прямой, или зигзагом, быстро или медленно), он для каждого своего положения находит ближайшую точку дороги, и проверяет её в графе. У графа есть поле, которое определяет минимальное расстояние между узлами (то есть если у нас в графе есть какая-то точка, и мы пытаемся добавить в граф точку, которая на находится на меньшем расстоянии, чем минимальное, то эта точка не добавится в граф). После того, как бот проедет карту и вернётся туда, откуда начал, у граф карты будет готов. Далее граф загружается в класс “GraphDrawer”, и рисуется изображение (из библиотеки OpenCV).

* 1. **Практическая применимость**

Этот проект применим в симуляторе “gym-DuckieTown”, например, карты дорог, полученные с его помощью, можно использовать как эталон того, как должны выглядеть карты дорог, чтоб помогать тем, кто будет пытаться написать программы для составления карты дорог.

Кроме этого, этот проект может применяться не в симуляторе, а на реальных дорогах (если его адаптировать, разумеется). К примеру, машина, которая не может получить карту той местности, где она находится, сможет в ней ориентироваться, чтобы не попасть в ДТП.

1. **Перспективы развития проекта**

Можно бесконечно улучшать класс “AutoDriver”, чтоб бот ездил ещё более плавно; добавить распознавание перекрёстков и сделать так, чтобы бот объездил всю карту (ведь текущая версия составляет карту дорог не полностью, если на карте есть перекрёстки, поскольку бот, пока что, не умеет на перекрёстке выбирать направление движения).

Также можно адаптировать бота под реальные дороги, реальные машины; перспективы развития тут просто огромны, ведь сейчас активно развиваются автомобили, движущиеся почти без помощи человека.