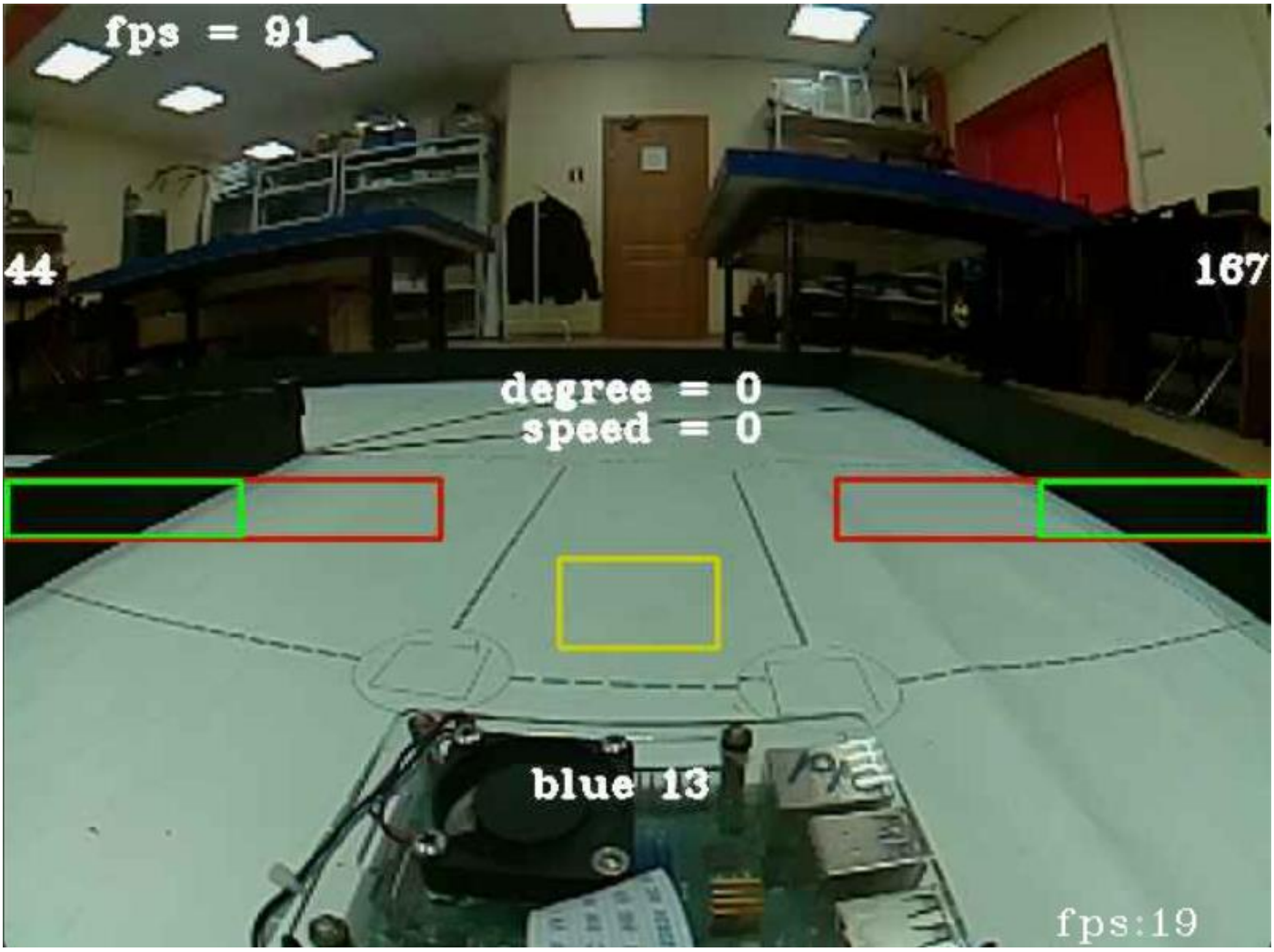
Обсуждение систем распознавания препятствий

Квалификация

Для начала следует взглянуть на поле глазами робота



На этом фото мы можем видеть поле, телеметрию (текст на экране), а также прямоугольники. Эти прямоугольники — это графическое отображения областей интереса. Они очерчивают границы области поиска тех или иных объектов, например, два красных прямоугольника очерчивают границы областей поиска бортиков, а небольшой жёлтый прямоугольник по центру — область поиска синей и оранжевой линии. В дальнейшем для простоты и краткости область интереса и прямоугольник мы объединим в понятие датчик. То есть на кадре присутствует 3 датчика: 2 датчика бортика и один датчик перекрёстков (перекрёстком называется синяя или оранжевая линия, в зависимости от направления движения).

Датчики

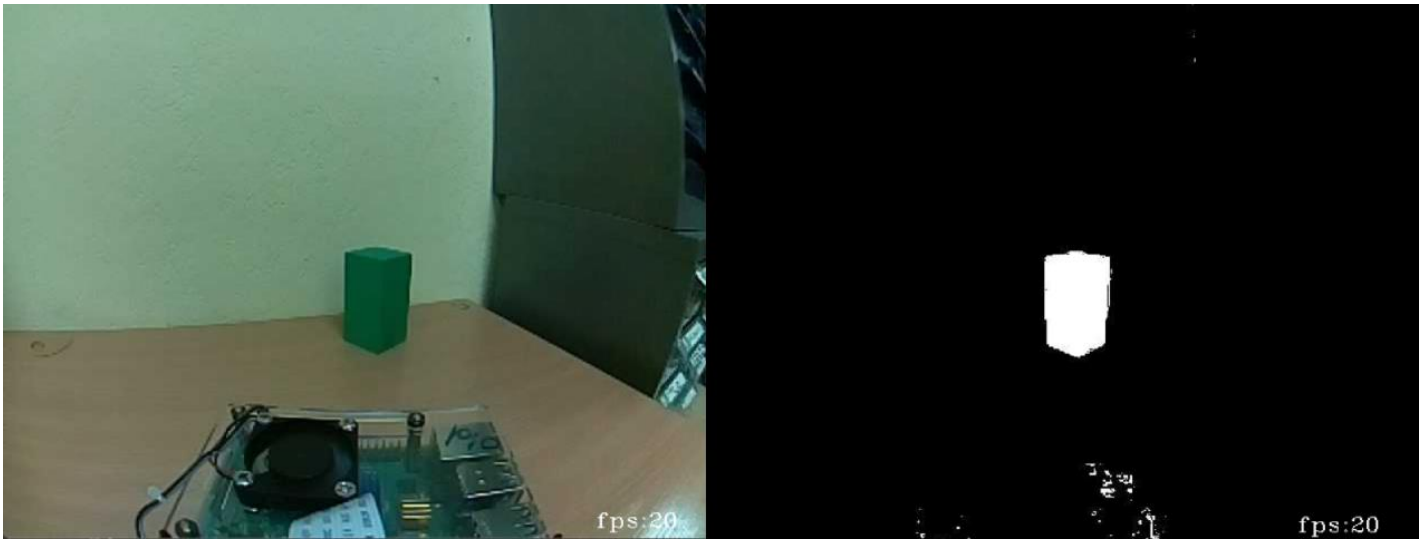
Датчики нужны для того, чтобы определить позицию объекта (бортика, знака, линии) по цвету в модели HSV. Все датчики устроены похожим образом и имеют базовый набор действий:

1)Скопировать часть кадра 2) перевести полученное изображение в цветовую модель HSV 3) создать маску 4) найти на этой маске контуры 5) найти наибольший контур 6) передать его координаты.

Создание маски — это процесс создания массива нулей и единиц на основе изображения в цветовом пространстве HSV и диапазона значений. Алгоритм создания таков:

Если цвет пикселя на исходном изображении входит в данный диапазон, то пиксель с такими же координатами на маске становится белым, иначе пиксель будет чёрным.

Демонстрация создания маски



Датчики бортиков

Их следует рассмотреть подробнее, так как они работают не совсем так, как работают датчики перекрёстков или знаков. На самом деле в каждом из датчиков бортика 20 маленьких датчиков, и каждый из них возвращает ноль или единицу, нет бортика и есть бортик соответственно. До этого показаниями датчиков являлись координаты левого и правого краёв наибольшего контура, а теперь это количество датчиков, содержащих контур бортика, проходящий по размеру. Это нововведение позволило нам снизить дискретность датчиков с 220 до 20, а следовательно, убрать большинство шумов и сделать движение машинки более плавным.

(вставить фотку)

Алгоритм движения

После разбора основных составляющих можно перейти к общему алгоритму движения, он будет представлен ниже в форме псевдокода.

```
Получить кадр с камеры

Если робот находится в фазе движения:
      Обработать датчики бортиков
      ПД регулятор
      Обработать датчик перекрёстка

Если робот находится в фазе активного торможения:
      Если с перехода в фазу активного торможения прошло менее 100 миллисекунд:
            скорость = -100
      Иначе:
            перейти в фазу остановки

Если робот находится в фазе остановки:
      Скорость = 0
      Угол поворота = 0

Если количество перекрёстков равно 12 и флаг опущен:
      Поднять флаг
      Засечь время
      Количество перекрёстков = 13

Если с момента 12-го перекрёстка прошло больше времени чем нужно для преодоления половины зоны:
      Перейти в фазу активного торможения
      Засечь время
      Опустить флаг

Передать значение угла поворота из ПД регулятора на микроконтроллер
```

В данном фрагменте упущены некоторые части кода, не влияющие на прохождение траектории, такие как индикация или подсчёт кадров в секунду.

Но если рассматривать более подробно, то для начала следует посмотреть на код программы.

```
wait_for_key()
while 1:
    frame = robot.get_frame(wait_new_frame=1)

    if state == 1:
        red, green, blue = 0, 0, 0
        detect_line_pro()
        pd_regulator(dat1, dat2)
        search_cross()
        if search_cross_time + 0.5 > time.time():#условие зажигания светодиода
            if color_line == 'orange':
                red = 84
                green = 54
            if color_line == 'blue':
                blue = 80
                green = 20
```

Этот код мы взяли из программы qualification.py и убрали все комментарии. Тут, очевидно, не вся программа, это только фаза движения, которая, в прочем, представляет собой весь алгоритм.

`wait_for_key()` — это функция ожидания нажатия кнопки, она посылает на микроконтроллер сообщение '9999999999999999999$', которое свидетельствует о том, что распберри запустилась, и начала выполнять программу. Параллельно происходит чтение сообщений с микроконтроллера, которые содержат всего одну цифру, ноль либо единицу. Ноль — кнопка не нажата, один — кнопка нажата. Как только приходит сообщение о нажатой кнопке, распберри выходит из этой функции.

`frame = robot.get_frame(wait_new_frame=1)` — этой строчкой мы получаем снимок с камеры, который мы будем использовать в дальнейшем.

`if state == 1:` — переменная state отвечает за режимы работы робота, 1 — выполнение задания, 2 — активное торможение, 3 — остановка.

`red, green, blue = 0, 0, 0` — эти три переменные отвечают за цвета RGB светодиода.

`detect_line_pro()` — эта функция отвечает за работу датчиков бортиков, но не отрисовывает их на экране. Внутри у неё не так много кода, потому что она использует две другие функции и упрощает работу с ними. Её задача просто приравнять глобальные переменные dat1 и dat2, к показаниям датчиков.

`pd_regulator(dat1, dat2)` — эта функция обыкновенный пропорционально-дифференциальный регулятор с некоторыми дополнениями в виде частных случаев, в роде: если левый датчик равен нулю, следовательно мы на повороте и только что проехали угол, следовательно сейчас необходимо сделать поворот влево.

`search_cross()` — эта функция занимается поиском перекрёстков на соответствующем датчике. Эта же функция записывает время зон в список.

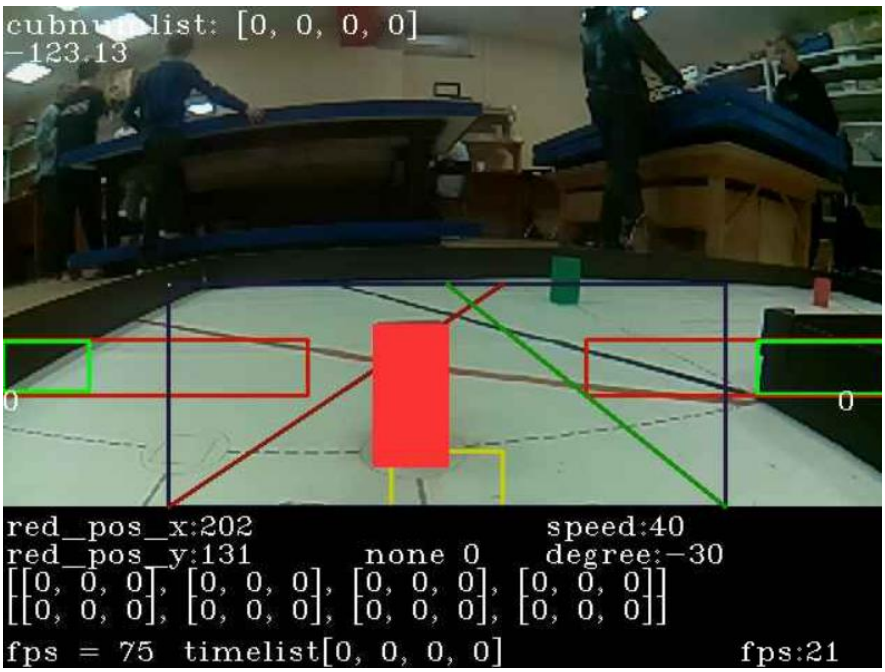
А далее идет индикация обнаружения перекрёстков.

И это весь алгоритм движения, дальше только активное торможение и остановка.

Финал

Финальный заезд отличается от квалификационного только знаками на поле, соответственно, программа будет отличаться только алгоритмами, связанными со знаками.

Знаки мы определяем по HSV с помощью датчика, как и остальные объекты.



Экран финального заезда отличается от квалификационного только датчиком знаков. Так как знаки могут быть где угодно и на любом расстоянии от робота, датчик должен быть довольно большим. Когда робот замечает знак, он начинает ориентироваться только по нему, игнорируя бортики. Для этого у нас есть отдельная функция с ПД регулятором (pd_regulator_cube). Эта функция отличается от обычного ПД регулятора тем, что в ней ошибка высчитывается исходя из перспективы, которая изображена зелёной и красной линиями на экране. Перспектива подразумевает, что в зависимости от расстояния до знака мы должны занимать разные позиции в пространстве относительно него. Например, если красный знак на камере высоко, то он достаточно далеко от робота (рис. 1), и чтобы к нему правильно подъехать, нужно фиксировать его по середине кадра, а если он низко на камере, то есть находится возле робота (рис. 2), то, чтобы его не сбить, робот должен находиться максимально справа.

Рис. 1

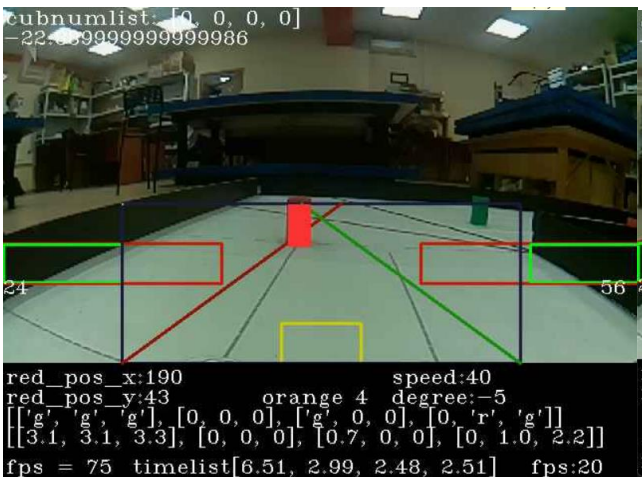
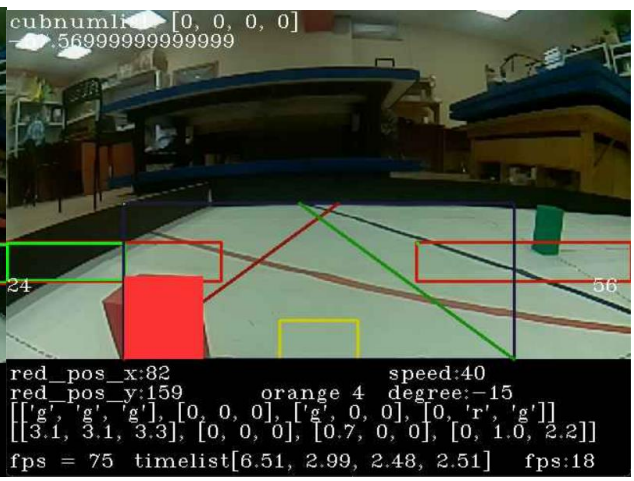


Рис. 2



Алгоритм фазы движения финального заезда выглядит так:

```
Обработать датчики знаков
Если ни одного знака не обнаружено:
    Обработать датчики бортиков
    Двигаться с помощью ПД регулятора
Если кубик, который нужно объезжать с внешней стороны, пропал менее 400 миллисекунд назад:
        Совершить поворот в сторону движения
        #это сделано для того, чтобы быстро вернуться на траекторию
Иначе если есть зеленый и он ближе:
    Двигаться с помощью ПД регулятора для знаков по зелёному знаку
Иначе
    Двигаться с помощью ПД регулятора для знаков по красному знаку
Обработать датчик перекрёстков
```

Фаза активного торможения в финальном заезде отсутствует, так как финальный заезд проходится с меньшей скоростью, а фаза остановки абсолютно идентична.

Далее будет представлен фрагмент кода, и более подробный его разбор.

```
if state == 1:  # езда
    red, green, blue = 0, 0, 0
    if search_cross_time + 0.5 > time.time():
        if color_line == 'orange': # если цвет перекрёстка оранжевый зажечь оранжевый
            red = 60
            green = 50
        if color_line == 'blue':  # если цвет перекрёстка синий зажечь синий
            blue = 80
            green = 20
    green_pos = cube_g()
    red_pos = cube_r()
    green_pos_x = green_pos[0] - 1
    green_pos_y = green_pos[3]
    red_pos_x = red_pos[2] + 1
    red_pos_y = red_pos[3]

    if green_pos_x == 0 and red_pos_x == 0:  # если нет кубиков
        detect_line_pro()  # ищем бортики
        pd_regulator(dat1, dat2, kp, kd)

        if cube_green_exist + 0.4 > time.time() and color_line == 'orange':
            degree = -50
        if cube_red_exist + 0.4 > time.time() and color_line == 'blue':
            degree = 50

    elif green_pos_x > 0 and green_pos_y > red_pos_y:
        cube_color = "Green"
        cube_exist_tim = time.time()
        if color_line == 'blue':
            b_g = 150
        elif color_line == 'orange':
            b_g = 180
        else:
            b_g = 160
        pd_regulator_cube(green_pos_x, (b_g + green_pos_y * 1.23))
        red, green, blue = 0, 100, 20

    else:  # если есть красный
        cube_color = "Red"
        cube_exist_tim = time.time()
        if color_line == 'blue':
            b_r = 230
        elif color_line == 'orange':
            b_r = 220
        else:
            b_r = 240
        pd_regulator_cube(red_pos_x, (b_r - red_pos_y * 1.23))
        red, green, blue = 100, 0, 0

    search_cross()
```