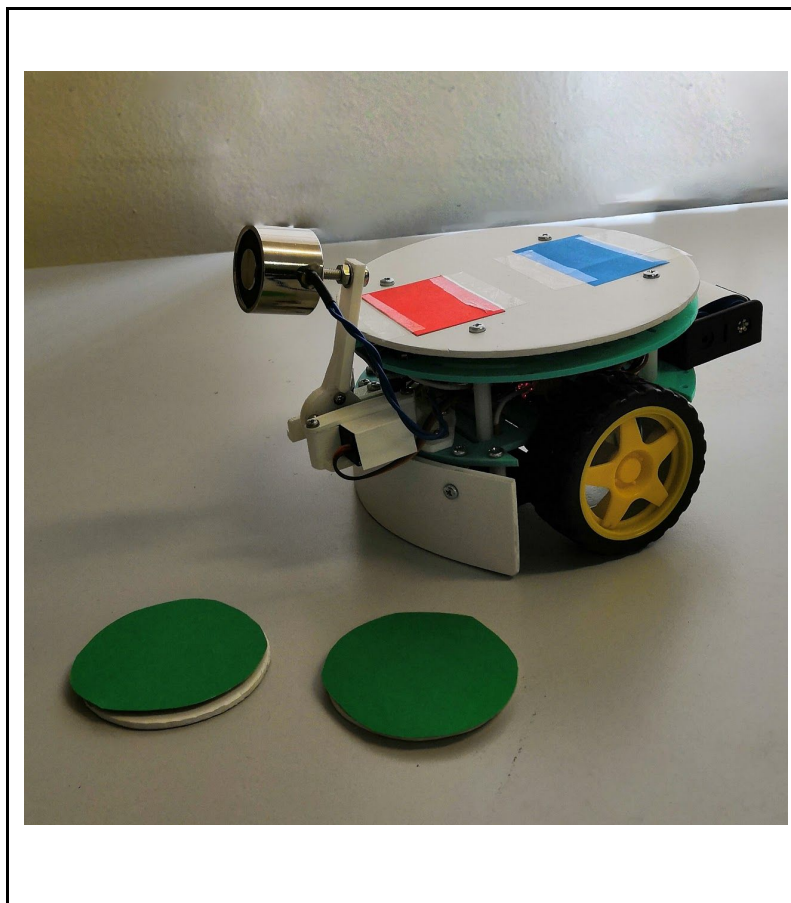


ЦТПО МИЭТ

## ТЕХНИЧЕСКОЕ ОПИСАНИЕ УСТРОЙСТВА

Автономная система навигации робота в поле зрения камеры на основе  
Raspberry Pi3B



Задача № 4

Выполнили:

Школа № 1557

Цатурьян Константин Артурович

Довгаль Даниил Станиславович

Воробьёв Никита Александрович

Сарибекян Гор Эдикович

Шугаев Максим Павлович

Москва 2019

## **Содержание**

1. Условие задачи и техническое задание
2. Описание работы решения
3. Структурно-функциональная схема
4. Конструкционные особенности
5. Алгоритм работы
6. Приложение

## Постановка ТЗ

### 1. Условия задачи.

Спроектировать и реализовать конструкцию мобильного робота, перемещающегося три предмета, произвольно расставленных на полигоне, в любой из углов данного полигона, откуда был произведен запуск робота. Для автономного решения задачи навигации - определения угловой ориентации робота, его координат и координат перемещаемых им предметов – необходимо использовать видеокамеру, стационарно закрепленную над полигоном.

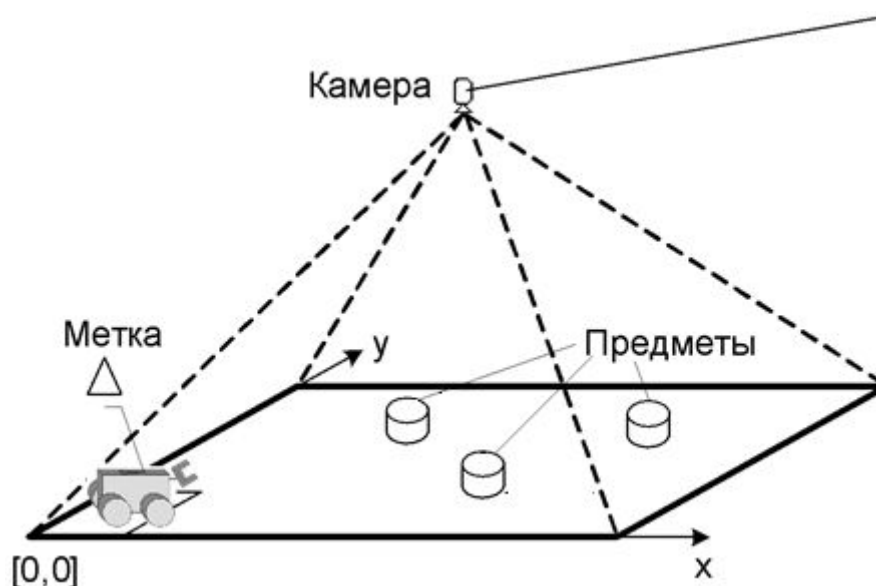
### 2. Техническое задание

Для навигации робота необходимо установить над полигоном видеокамеру, которая позволит определять координаты робота  $[x, y]$  по считанной с верхней части робота контрастной метке. На поверхности предметов также наносятся контрастные метки, которые должны отчетливо распознаваться видеокамерой. Они используются для навигации при перемещении робота по поверхности полигона. Распознанные и идентифицированные с помощью камеры, подвешенной над полигоном, коды контрастной метки однозначно задают ориентацию и положение робота. Координаты предметов также считываются по нанесенным на них контрастным меткам с помощью той же камеры. Для перемещения предметов необходимо оснастить робота системой захвата (например, магнитной).

Состав датчиков и необходимая аппаратная и программная комплектация робота определяется участниками с учетом излагаемых организаторами рекомендаций.

В составе робота не должно быть датчика сканирования препятствий. Запрещено наносить невидимую разметку и применять оптические проекции и любые подсветки на полигон.

Тестовый полигон (см. рисунок ниже) представляет собой участок (материал участка любой, например фанера, ватман, линолеум, плитка и пр.) размером не более 1,5 на 1,5 метра, без ограждения и какой либо нанесенной внутри разметки, в том числе скрытой или спроецированной. Предметы для перемещения произвольны по форме, весу и материалу.



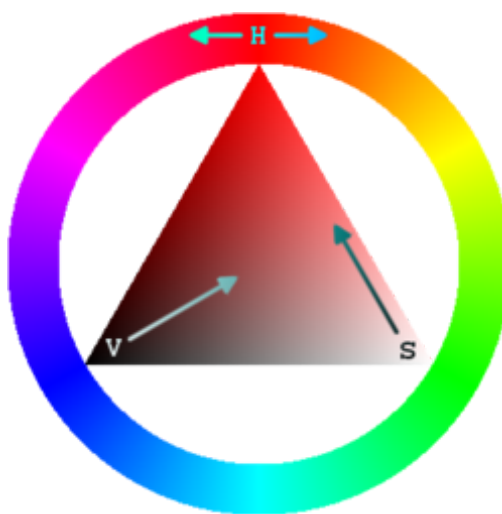
При установке стационарной видеокамеры над полигоном, необходимо снабдить ее направленным источником монохроматического света, осуществляющего подсветку полигона. Установка проводной связи видеокамеры с компьютером (ноутбуком), а также ее закрепление над полигоном осуществляется участниками. При этом связь ноутбука с роботом осуществляется по беспроводному каналу передачи данных. После вычисления координат расставленных предметов осуществляется вычисление траектории перемещения до них мобильного робота.

В каждой точке траектории движения робота, между установленными предметами и началом координат, его положение и ориентация определяются путем многократного считывания и обработки видеоизображения с камеры. В этих точках происходит вычисление, уточнение и корректировка отклонений от вычисленных траектории между началом координат и позициями из которых необходимо забрать предметы и перевезти их в стартовую позицию.

## Описание работы решения

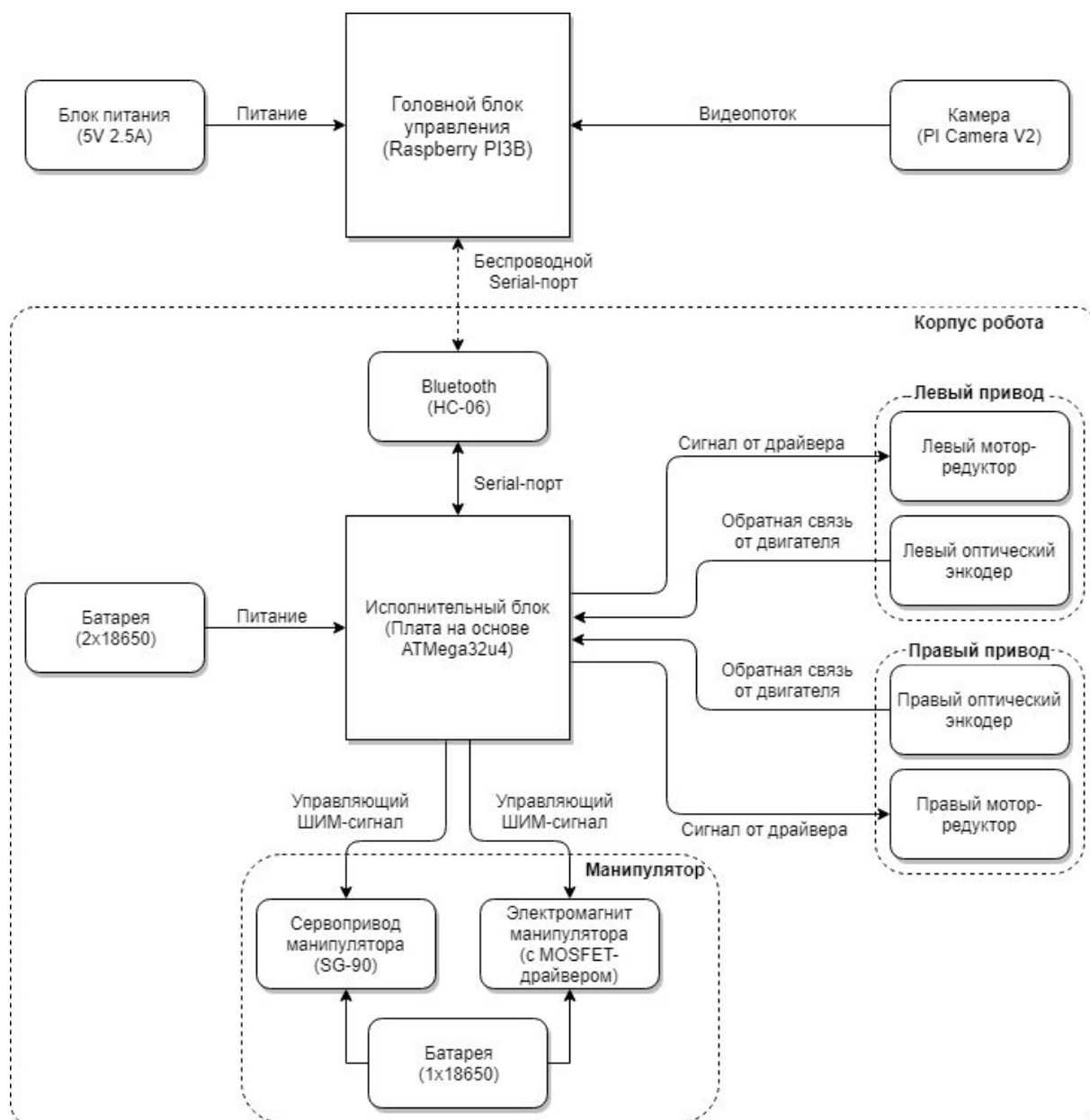
Для решения поставленной задачи был собран робот с манипулятором, в качестве полигона выбран белый лист размера А0 (ватман), собрана вертикальная стойка-штатив для крепления источника освещения, камеры и головного блока управления (Raspberry PI 3B) над полигоном.

В самом начале работы производится калибровка системы, которая включает в себя процедуры уточнения границ полигона, калибровка цветowych шаблонов для нахождения цветных маркеров на роботе и на трех шайбах, т.к. распознаваемые цвета могут измениться при другом освещении (см. Приложение). Калибровка цветов производится в цветовом пространстве HSV.



После калибровки, система готова к работе. Робот ставится в один из углов, и небольшая область вблизи угла полигона становится базой, на которую робот будет по очереди перетаскивать обнаруженные на полигоне объекты, начиная с наиболее близкого к базе, заканчивая самым дальним. В конце работы головной блок управления сообщит о завершении работы алгоритма.

## Структурно-функциональная схема




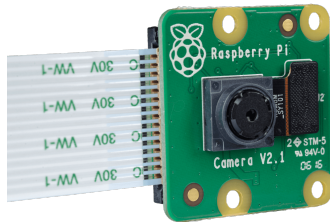
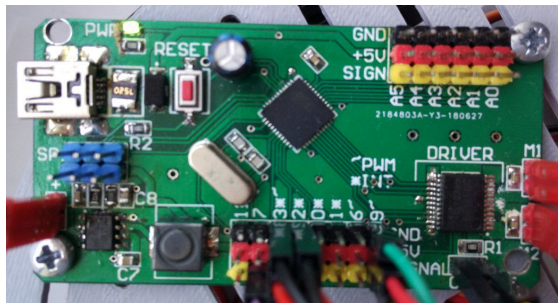


Система включает в себя 2 блока: головной блок управления (Raspberry Pi 3B) и исполнительный блок - робот с магнитным манипулятором на базе arduino. Блоки осуществляют передачу управляющих сигналов друг другу через bluetooth.

Головной блок представляет собой одноплатный компьютер Raspberry PI 3B и подключенную к нему камеру Pi Camera V2. Для реализации функции распознавания объектов было принято решение воспользоваться связкой python3 + OpenCV (библиотека для широкого анализа изображений).






Исполнительный блок есть робот на базе arduino-подобной платы, которая уже имеет на себе драйвер двигателей. Робот также включает в себя три аккумулятора форм-фактора 18650, саму плату, сервопривод для управления манипулятором,

электромагнит, bluetooth модуль hc-06, два мотор-редуктора, и оптические энкодеры для обратной связи от двигателей.

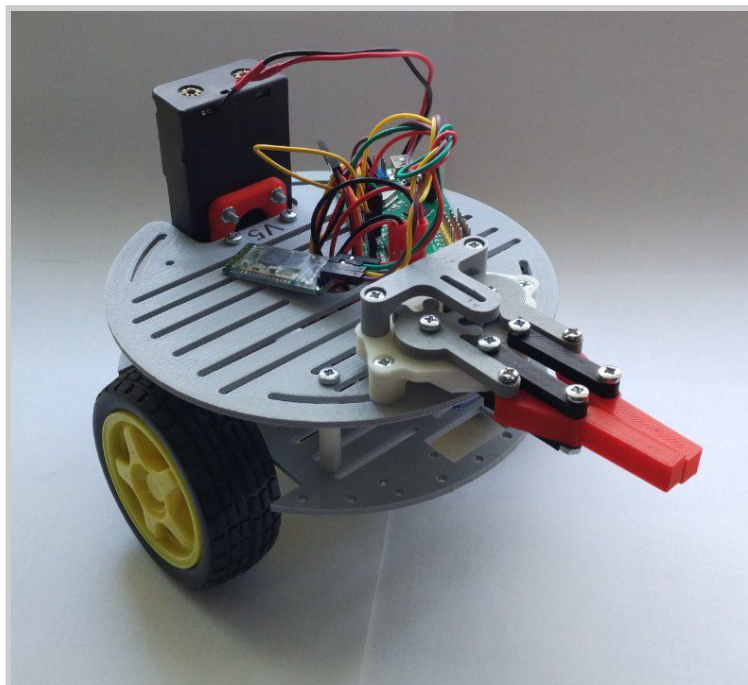
**Точный перечень используемых компонентов:**

Описание	Изображение
Одноплатный компьютер raspberry PI 3B Оснащен Bluetooth 4.0	
Камера PI Camera V2 8MP	
Плата управления роботом <ul style="list-style-type: none"> <li>основана на микроконтроллере ATmega32u4 (является аналогом Arduino leonardo)</li> <li>имеет на себе двухканальный драйвер двигателей, удобную развязку питания для подключения различных модулей</li> </ul>	
Bluetooth-модуль HC-06 <ul style="list-style-type: none"> <li>работает только в slave режиме</li> <li>работает через последовательный порт</li> </ul>	
Колесо 65 мм для робототехники совмещенное с мотор-редуктором <ul style="list-style-type: none"> <li>Диаметр колеса 65 мм;</li> <li>Мотор-редуктор 1: 48;</li> <li>Напряжение питания 3...6 В;</li> <li>Скорость вращения 180 об/мин при 6 В</li> </ul>	

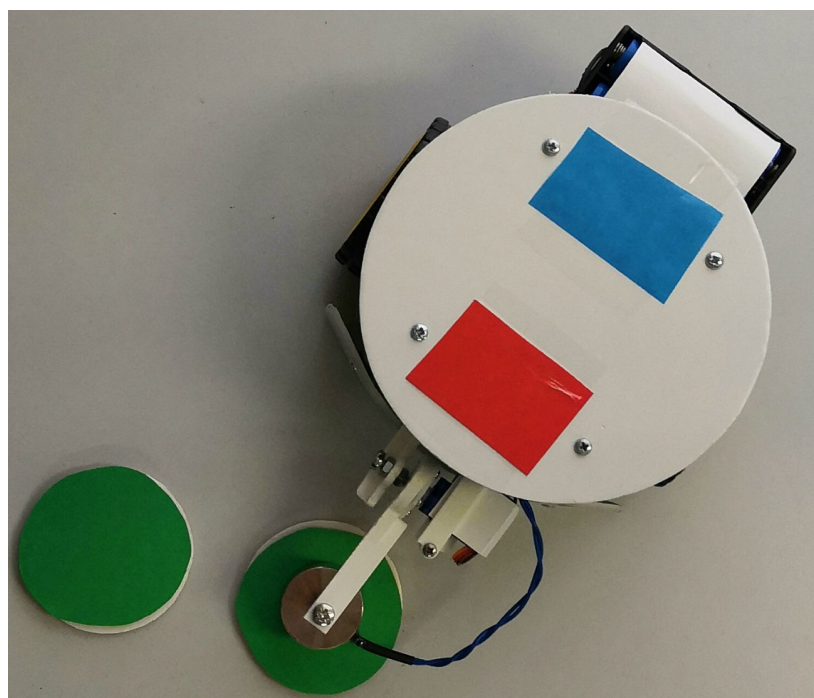


<p>Фотопрерыватель на плате. На выходе имеет TTL уровни 5 В, что позволяет подключать его напрямую к пинам Arduino</p>	
<p>Электромагнит</p> <ul style="list-style-type: none"> <li>• Номинальное напряжение: 5В</li> <li>• Потребляемый ток: до 0.35А</li> <li>• Усиление на отрыв: до 15 кг</li> <li>• Габариты: Ø20x15 мм</li> <li>• Крепежное отверстие: под винт М4</li> <li>• Вес электромагнита: 25 г;</li> </ul>	
<p>Сервопривод TowerPro SG90 с пластиковым механизмом. Рабочее напряжение 3.5 - 8.4В Общий вес 9г</p>	
<p>Кейс для одного и двух аккумуляторов 18650</p>	
<p>Аккумулятор 18650 3.7В (3шт.)</p>	

## Конструкционные особенности

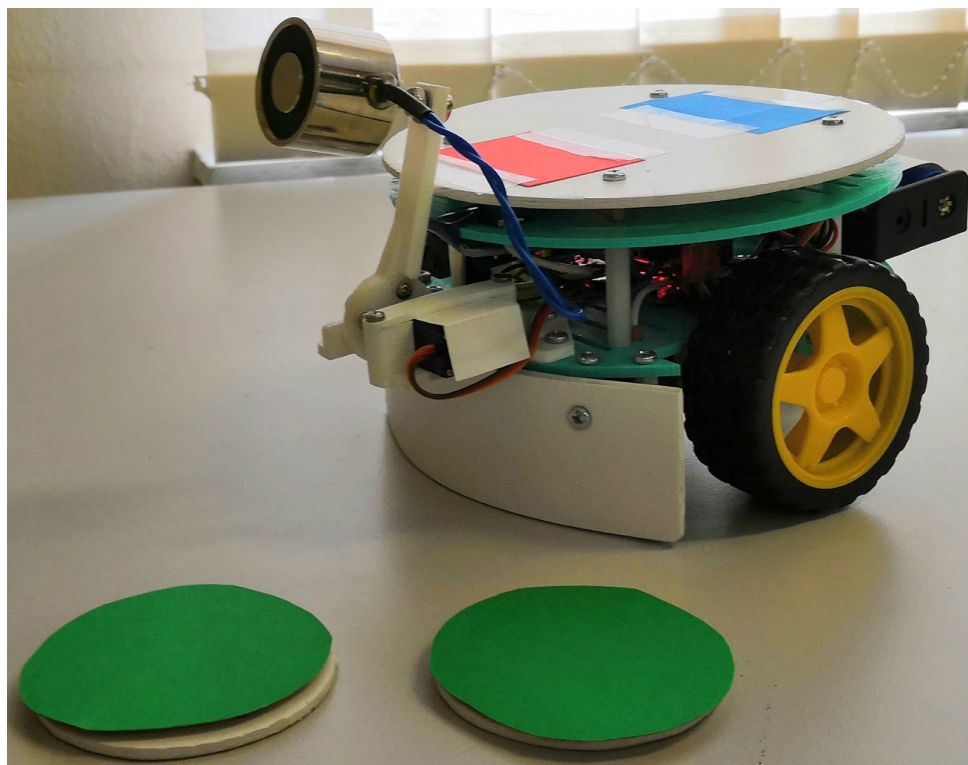


На рисунке выше изображен первичный прототип робота предоставленный МИЭТ'ом, на котором производились тесты программы. В дальнейшем этот робот был доработан под наши нужды и стал основой финального робота.



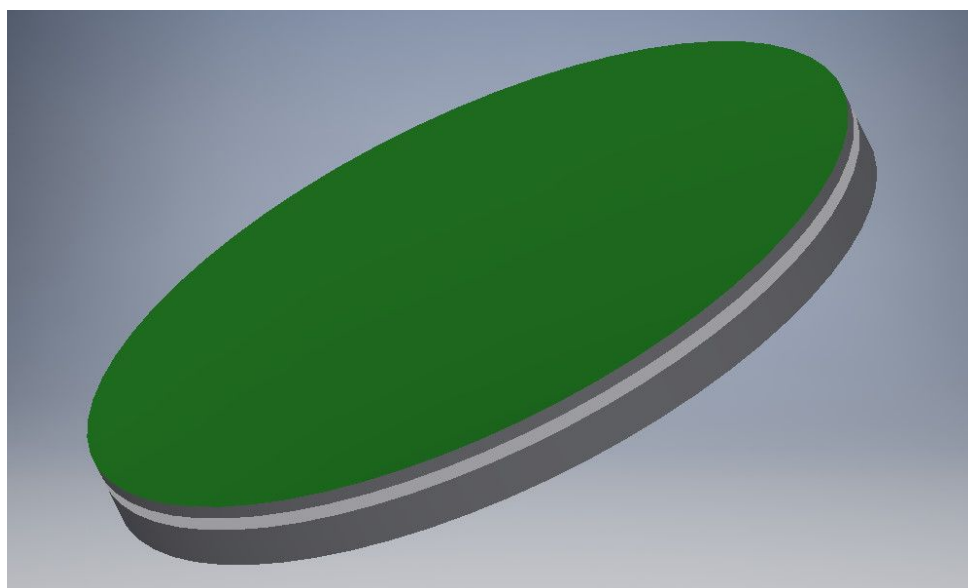
Сверху робота расположена круглая пластина с двумя маркерами: синим и красным, синий - задняя часть робота, красный - передняя. Используются именно два маркера для определения угла поворота робота. Белая пластина закрывает зелёный корпус от камеры, для избежания определения ненужных объектов.

Батарея и большинство внутренних элементов были смещены назад, из-за тяжёлого электромагнита для поддержания равновесия.



Спереди расположен щиток который не даёт роботу наехать на объекты и прикрывает фотопрерыватель.

Модуль-манипулятор представляет собой шарнир на сервоприводе с электромагнитом на конце для захвата шайб.

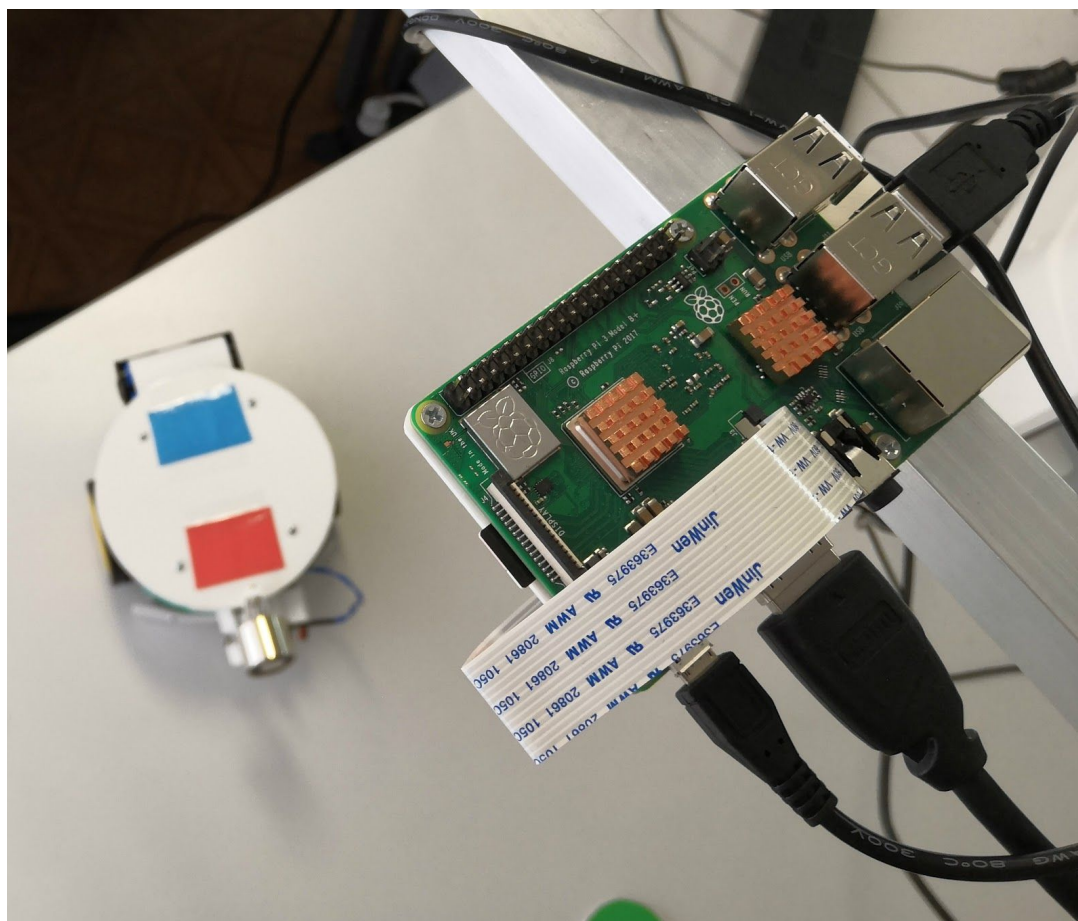
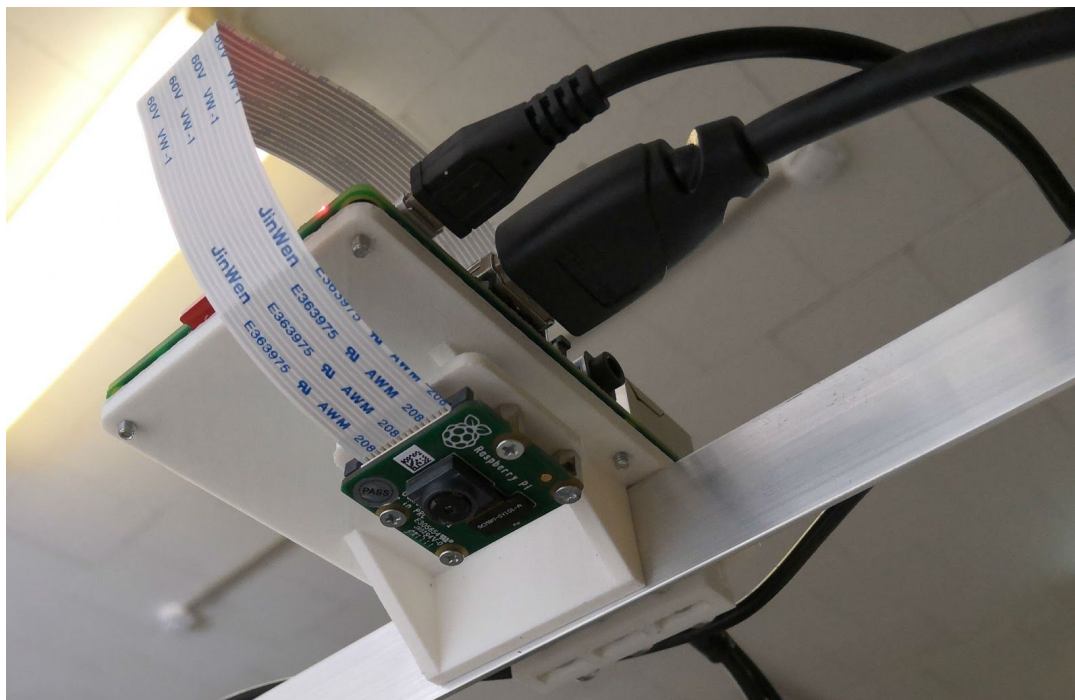


Шайбы имеют радиус 30 мм, изготавливаются на 3D-принтере, имеют внутри себя металлическую пластинку, за счет которой осуществляется примагничивание шайбы.

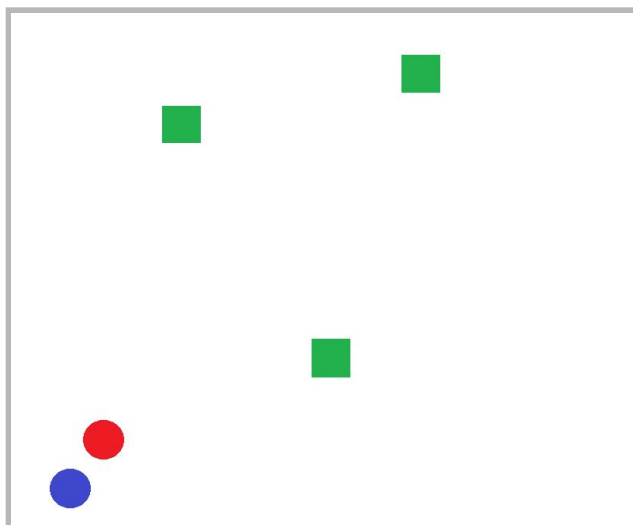


Шайбы покрываются стикерами зеленого цвета, чтобы камера была способна их обнаружить как контрастный яркий объект на белом листе ватмана.

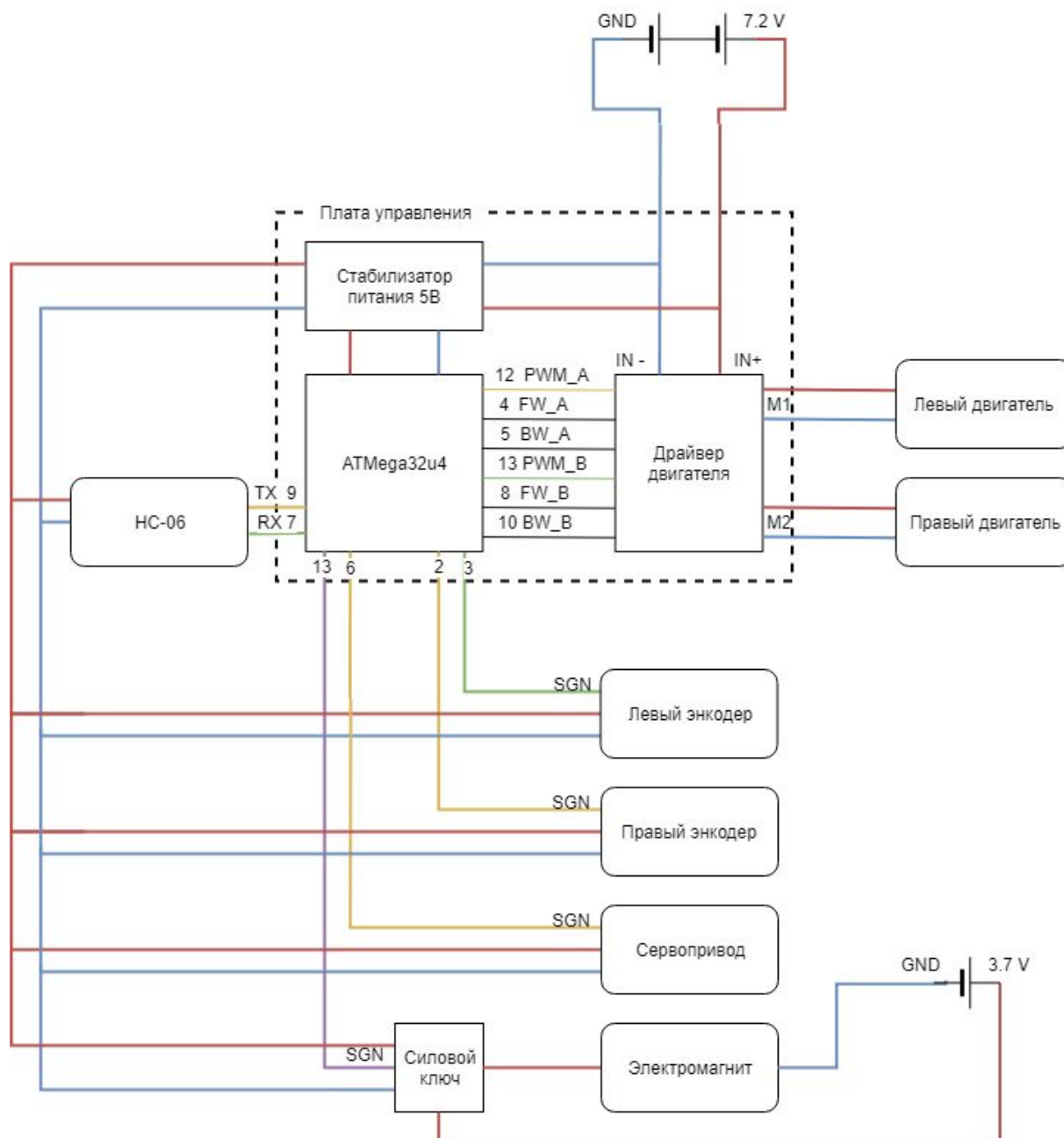
Шайбы регистрирует камера установленная на штативе над полигоном.



Микрокомпьютер подключен к электросети, мышке, клавиатуре и монитору для запуска, отладки, калибровки программ и камеры, наблюдения за процессом выполнения программы.



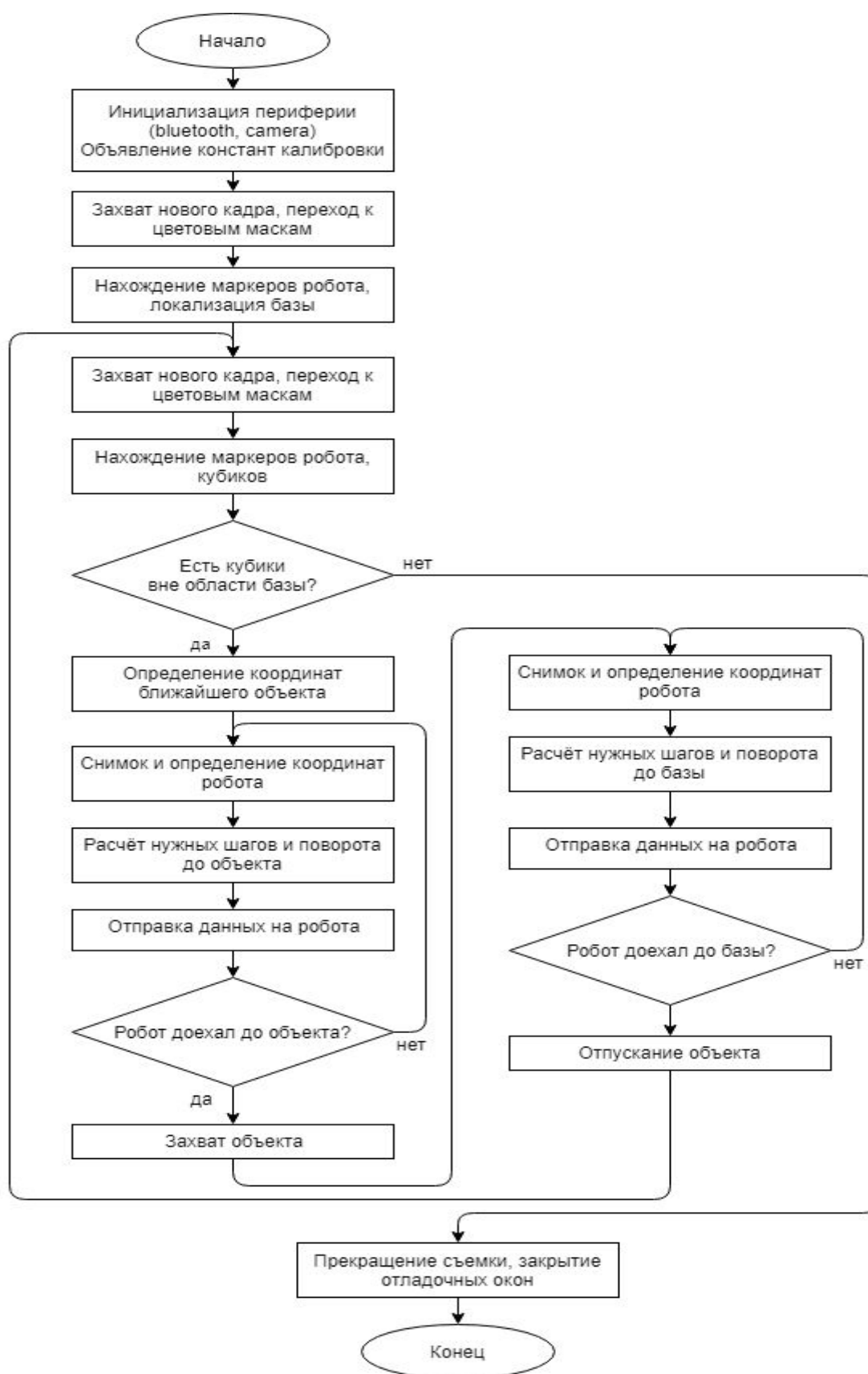
На рисунке выше представлено идеальное изображение, примерно так выглядит полигон сверху.

**Схема соединения компонентов робота:**

На изображении схемы соединения обозначены номера пинов и обозначения, красный и синий - провода питания (+ и - соответственно).

# Алгоритм работы

## 1. Головной блок



Выше представлен упрощенный алгоритм работы системы, детали блоков намеренно опущены (полный код скрипта см. Приложение, Листинг 2), основная суть алгоритма сохранена, каждый блок снабжен исключением, выводящим в консоль причину ошибки. Это позволяет быстро обнаружить и устранить неполадку.

Алгоритм работы подразумевает следующий цикл: делаем кадр - производим исполнение алгоритмов обнаружения контуров цветных объектов - находим все нужные объекты (робот, 3 шайбы) - производим проверку, есть ли хоть одна шайба вне окрестности базы, если есть, то прокладываем кратчайший путь до ближайшей шайбы, если шайб вне базы нет, то алгоритм завершает работу. Если шайба вне окрестности базы обнаружена и проложен маршрут, то производится дискретное движение робота (пересылаются команды перемещения на определенное расстояние вперед/назад, поворот на определенный угол влево/вправо), затем, при получении с робота сигнала о завершении своего движения, производится еще один снимок, снова оценивается положение и так далее, пока робот не приблизится к шайбе на нужное для ее захвата расстояние. По достижению шайбы осуществляется захват манипулятором. Аналогичным образом малыми перемещениями робот достигает базы и сбрасывает шайбу. Если все объекты лежат внутри базы, то работа программы прекращается.

## 2. Исполнительный блок

8 bit	8 bit
<b>opcode</b>	<b>value</b>

Программа исполнительного блока (см. Приложение, Листинг 3) построена следующим образом: с bluetooth-serial порта принимаются 2 байта, первый - “opcode” (код операции), второй - “value” (передаваемое значение). Это универсальная, проверенная временем схема, позволяющая кодировать операции. После приема информации в зависимости от кода операции, робот движется вперед на n шагов, где n - принятое значение value, поворачивает влево/вправо на n шагов, и т.д. Программа разработана таким образом для последующего легкого наращивания функциональных возможностей. Также робот сообщает головному блоку информацию о завершении каждого действия, например, при окончании движения отсылается контрольный байт, символизирующий готовность к приему следующей команды. Головной блок ждет этот контрольный байт, после чего может продолжать работу алгоритма. В силу



особенностей работы serial-порта, принимаемые байты по bluetooth скапливаются в буфер-очередь, поэтому можно отослать пакет команд, которые будут исполняться последовательно. Это нужно, например, для набора скорости робота и т.д.

Расшифровка пакетов:

<b>opcode (HEX) + value</b>	<b>Расшифровка</b>
0x10 0x(value)	движение назад на <b>value</b> шагов
0x11 0x(value)	движение вперед на <b>value</b> шагов
0x20 0x(value)	поворот влево на <b>value</b> шагов
0x21 0x(value)	поворот вправо на <b>value</b> шагов
0x30 0x(value)	повернуть манипулятор на угол <b>value</b>
0x40 0x00	выключить электромагнит
0x40 0xFF	включить электромагнит

0x33	‘!’ - контрольный байт
------	------------------------

## Приложение

Листинг 1. Скрипт для калибровки цветowych масок

```
import cv2
import numpy as np

if __name__ == '__main__':
    def nothing(*arg):
        pass

cv2.namedWindow("result") # создаем главное окно
cv2.namedWindow("settings") # создаем окно настроек

# Захват видео с камеры
cap = cv2.VideoCapture(0)

# Разрешение видео
cap.set(3,640) # Ширина
cap.set(4,480) #Высота

# создаем 6 бегунков для настройки начального и конечного цвета фильтра
cv2.createTrackbar('h1', 'settings', 0, 180, nothing)
cv2.createTrackbar('s1', 'settings', 0, 255, nothing)
cv2.createTrackbar('v1', 'settings', 0, 255, nothing)
cv2.createTrackbar('h2', 'settings', 0, 180, nothing)
cv2.createTrackbar('s2', 'settings', 0, 255, nothing)
cv2.createTrackbar('v2', 'settings', 0, 255, nothing)

while True:
    flag, img = cap.read()
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # считываем значения бегунков
    h1 = cv2.getTrackbarPos('h1', 'settings')
    s1 = cv2.getTrackbarPos('s1', 'settings')
    v1 = cv2.getTrackbarPos('v1', 'settings')
    h2 = cv2.getTrackbarPos('h2', 'settings')
```

```
s2 = cv2.getTrackbarPos('s2', 'settings')
v2 = cv2.getTrackbarPos('v2', 'settings')

# формируем начальный и конечный цвет фильтра
h_min = np.array((h1, s1, v1), np.uint8)
h_max = np.array((h2, s2, v2), np.uint8)

# накладываем фильтр на кадр в модели HSV
color_mask = cv2.inRange(hsv, h_min, h_max)

cv2.imshow('result', color_mask)

ch = cv2.waitKey(20)
if ch == 27:
    break

cap.release()
cv2.destroyAllWindows()

cv2.imshow('result', color_mask)
ch = cv2.waitKey(20)
if ch == 27:
    break
cap.release()
cv2.destroyAllWindows()
```

## Листинг 2. Главная программа управления

```
import numpy as np
import cv2
import math
import serial

def determine_base(x, y, a, w, h):
    if x - a <= 0: x1 = 0
    else: x1 = x - a

    if x + a >= w: x2 = w
    else: x2 = x + a

    if y - a <= 0: y1 = 0
    else: y1 = y - a

    if y + a >= h: y2 = h
    else: y2 = y + a

    return [int(x1), int(x2), int(y1), int(y2)]

def detect_robot_pos(x1, y1, x2, y2):
    cx = (x1 + x2) / 2
    cy = (y1 + y2) / 2
    angle = math.degrees(math.atan2((y2 - y1), (x2 - x1)))

    return cx, cy, angle

def get_dist(r_x, r_y, r_angle, item_coords):
    dists = []
    for pair in item_coords:
        x = pair[0]
        y = pair[1]
        dists.append(math.sqrt((x - r_x)**2 + (y - r_y)**2))

    i_min = dists.index(np.amin(dists))
    min_dist = np.amin(dists)
    min_dist_coords = item_coords[i_min]

    o_x = min_dist_coords[0]
    o_y = min_dist_coords[1]

    required_angle = math.degrees(math.atan2((o_y - r_y), (o_x - r_x))) - r_angle
    if required_angle < -180:
```

```

        required_angle = required_angle + 360
    if required_angle > 180:
        required_angle = required_angle - 360

    return min_dist, required_angle, [o_x, o_y]

def detect_color_objects(colormask, accuracy = 50, take_all = False):
    cnts = cv2.findContours(colormask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[1]

    coords = []

    if take_all == True:
        for cnt in cnts:
            mmnt = cv2.moments(cnt)
            area = mmnt['m00']
            if area > accuracy:
                cx = int(mmnt['m10']/area)
                cy = int(mmnt['m01']/area)
                coords.append([cx, cy])

    else:
        cnt = max(cnts, key = cv2.contourArea)
        mmnt = cv2.moments(cnt)
        area = mmnt['m00']
        if area > accuracy:
            cx = int(mmnt['m10']/area)
            cy = int(mmnt['m01']/area)
            coords.append([cx, cy])

    return coords

def send_package(opcode, value):
    try:
        package = bytearray()
        package.append(opcode)
        package.append(value)
        bluetoothSerial.write(package)
        return True
    except:
        print('Error while sending package')
        return False

def wait_done_package():
    if bluetoothSerial.read() == b'!':

```

```

        print('Got OK!')
        return True
    else:
        print("There is no robot's feedback")
        return False

## settings ##

# bluetooth init
try:
    bluetoothSerial = serial.Serial("/dev/rfcomm0")
    bluetoothSerial.baudrate = 9600
    bluetoothSerial.bytesize = serial.EIGHTBITS
    bluetoothSerial.timeout = 10
except Exception:
    print('Error: Bluetooth init failure')
    exit()
else:
    print('Bluetooth init OK')

# camera init
WIDTH = 640
HEIGHT = 480

cap = cv2.VideoCapture(0)

cap.set(3, WIDTH) # Width
cap.set(4, HEIGHT) # Height
# cap.set(25, 10) # FPS
cap.set(38, 1) # Buffer size

#Red ranges
hsv_orange_min = np.array((0, 80, 60), np.uint8)
hsv_orange_max = np.array((20, 255, 255), np.uint8)

hsv_violet_min = np.array((150, 80, 60), np.uint8)
hsv_violet_max = np.array((180, 255, 255), np.uint8)

#Green ranges
hsv_green_min = np.array((40, 80, 40), np.uint8)
hsv_green_max = np.array((80, 255, 255), np.uint8)

#Blue ranges
hsv_blue_min = np.array((90, 120, 50), np.uint8)
hsv_blue_max = np.array((155, 255, 255), np.uint8)

```

```

close_area_0 = 145
close_area_1 = 200
wheel_const = 0.12
rotate_const = 0.17
angle_accuracy = 3
base_size = 80

## main ##

for i in range(3):
    cap.grab()
flag, image = cap.retrieve()

if flag == False:
    print('Error: Image is empty')
    exit()

# image = cv2.imread('test_img.png')

hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

orange_mask = cv2.inRange(hsv_image, hsv_orange_min, hsv_orange_max)
violet_mask = cv2.inRange(hsv_image, hsv_violet_min, hsv_violet_max)
red_mask = orange_mask + violet_mask
blue_mask = cv2.inRange(hsv_image, hsv_blue_min, hsv_blue_max)

r_coords = detect_color_objects(red_mask)[0]
b_coords = detect_color_objects(blue_mask)[0]

cv2.arrowsLine(image, (b_coords[0], b_coords[1]), (r_coords[0], r_coords[1]), (255, 0, 255),
5)

robot_x, robot_y, robot_angle = detect_robot_pos(b_coords[0], b_coords[1], r_coords[0],
r_coords[1])

base_x, base_y = robot_x, robot_y
base_limits = determine_base(robot_x, robot_y, base_size, WIDTH, HEIGHT)

cv2.rectangle(image, (base_limits[0], base_limits[2]), (base_limits[1], base_limits[3]), (0,
0, 255), 3)

cv2.imshow('result', image)

while True:
    if cv2.waitKey(500) == 32:
        break

```

```

find_item_state = True
move_to_base_state = False
EXIT_flag = False

while EXIT_flag == False:
    for i in range(3):
        cap.grab()
    flag, image = cap.retrieve()

    if flag == False:
        print('Error: Image is empty')
        exit()

    # image = cv2.imread('test_img.png')

    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    green_mask = cv2.inRange(hsv_image, hsv_green_min, hsv_green_max)

    g_coords = detect_color_objects(green_mask, 50, True)

    new_g_coords = [[x, y]
                     for x, y in g_coords
                     if not(base_limits[0] <= x <= base_limits[1] and base_limits[2] <= y <=
base_limits[3])]

    g_coords = new_g_coords

    # if len(g_coords

for pair in g_coords:
    cv2.circle(image, (pair[0], pair[1]), 5, (255, 255, 255), 2)

while find_item_state == True:
    for i in range(3):
        cap.grab()
    flag, image = cap.retrieve()

    if flag == False:
        print('Error: Image is empty')
        exit()

    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

```



```

orange_mask = cv2.inRange(hsv_image, hsv_orange_min, hsv_orange_max)
violet_mask = cv2.inRange(hsv_image, hsv_violet_min, hsv_violet_max)
red_mask = orange_mask + violet_mask
blue_mask = cv2.inRange(hsv_image, hsv_blue_min, hsv_blue_max)

r_coords = detect_color_objects(red_mask)[0]
b_coords = detect_color_objects(blue_mask)[0]

cv2.arrowsLine(image, (b_coords[0], b_coords[1]), (r_coords[0], r_coords[1]), (255,
0, 255), 5)

if len(g_coords) == 0:
    EXIT_flag = True
    print('Stop: There are no items out of base!')
    break

robot_x, robot_y, robot_angle = detect_robot_pos(b_coords[0], b_coords[1],
r_coords[0], r_coords[1])
dist, angle, target_coords = get_dist(robot_x, robot_y, robot_angle, g_coords)

cv2.circle(image, (target_coords[0], target_coords[1]), 8, (0, 0, 255), 2)

steps_move = int(dist * wheel_const)
steps_rotate = int(angle * rotate_const)

cv2.rectangle(image, (base_limits[0], base_limits[2]), (base_limits[1],
base_limits[3]), (255, 0, 255), 3)
cv2.imshow('result', image)

cv2.waitKey(1)

if math.fabs(steps_rotate) > angle_accuracy:
    if steps_rotate < 0:
        send_package(0x20, -steps_rotate)
        print('Rotate left ', -steps_rotate, ' steps')
    else:
        send_package(0x21, steps_rotate)
        print('Rotate right ', steps_rotate, ' steps')
else:
    if dist > close_area_0:
        if dist > close_area_1:
            if steps_move > 255:
                send_package(0x11, 0xff)
                print('Move forward 255 steps')
            else:
                send_package(0x11, int(steps_move*0.8))
                print('Move forward', int(steps_move*0.8), ' steps')

```

```

        else:
            send_package(0x11, 0x05)
            print('Move forward 5 steps')
    else:
        send_package(0x30, 0x30)
        if wait_done_package() == False:
            break
        send_package(0x40, 0xff)
        print('The item was taken!')
        find_item_state = False
        move_to_base_state = True

    if wait_done_package() == False:
        break

send_package(0x30, 0xb0)

if wait_done_package() == False:
    break

while move_to_base_state == True:
    for i in range(3):
        cap.grab()
        flag, image = cap.retrieve()

    if flag == False:
        print('Error: Image is empty')
        exit()

    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    orange_mask = cv2.inRange(hsv_image, hsv_orange_min, hsv_orange_max)
    violet_mask = cv2.inRange(hsv_image, hsv_violet_min, hsv_violet_max)
    red_mask = orange_mask + violet_mask
    blue_mask = cv2.inRange(hsv_image, hsv_blue_min, hsv_blue_max)

    r_coords = detect_color_objects(red_mask)[0]
    b_coords = detect_color_objects(blue_mask)[0]

    cv2.arrowedLine(image, (b_coords[0], b_coords[1]), (r_coords[0], r_coords[1]), (255,
0, 255), 5)

    robot_x, robot_y, robot_angle = detect_robot_pos(b_coords[0], b_coords[1],
r_coords[0], r_coords[1])
    dist, angle, target_coords = get_dist(robot_x, robot_y, robot_angle, [[base_x,
base_y]])

```

```

cv2.circle(image, (int(target_coords[0]), int(target_coords[1])), 8, (0, 0, 255), 2)

steps_move = int(dist * wheel_const)
steps_rotate = int(angle * rotate_const)

cv2.rectangle(image, (base_limits[0], base_limits[2]), (base_limits[1],
base_limits[3]), (255, 0, 255), 3)
cv2.imshow('result', image)

cv2.waitKey(1)

if math.fabs(steps_rotate) > angle_accuracy:
    if steps_rotate < 0:
        send_package(0x20, -steps_rotate)
        print('Rotate left ', -steps_rotate, ' steps')
    else:
        send_package(0x21, steps_rotate)
        print('Rotate right ', steps_rotate, ' steps')
else:
    if dist > close_area_0:
        if dist > close_area_1:
            if steps_move > 255:
                send_package(0x11, 0xff)
                print('Move forward 255 steps')
            else:
                send_package(0x11, int(steps_move*0.8))
                print('Move forward', int(steps_move*0.8), ' steps')
        else:
            send_package(0x11, 0x05)
            print('Move forward 5 steps')
    else:
        send_package(0x30, 0x30)
        if wait_done_package() == False:
            break
        send_package(0x40, 0x00)
        print('The item was put!')
        find_item_state = True
        move_to_base_state = False

    if wait_done_package() == False:
        break

send_package(0x30, 0xb0)

if wait_done_package() == False:

```

```
break
```

```
cv2.imshow('result', image)
```

```
if cv2.waitKey(500) == 27:
```

```
    break
```

```
print('Exiting...')
```

```
cap.release()
```

```
cv2.destroyAllWindows()
```

```
exit()
```

## Листинг 3. Программа исполнительного устройства.

```
#include <SoftwareSerial.h>
#include <Servo.h>

Servo manipulator;
SoftwareSerial bluetooth(9, 7); // RX, TX

const byte MB_PWM = 5;
const byte MB_F = 12;
const byte MB_B = 4;

const byte MA_PWM = 13;
const byte MA_F = 8;
const byte MA_B = 10;

const byte MAGNET = 11;

unsigned int stepsLeft = 0;
unsigned int stepsRight = 0;

void incrementStepsLeft()
{
    stepsLeft++;
}

void incrementStepsRight()
{
    stepsRight++;
}

void moveForward(byte steps, unsigned int &stepsLeft, unsigned int &stepsRight)
{
    stepsLeft = 0;
    stepsRight = 0;
    analogWrite(MA_PWM, 100);
    digitalWrite(MA_F, HIGH);
    digitalWrite(MA_B, LOW);

    analogWrite(MB_PWM, 100);
    digitalWrite(MB_F, HIGH);
    digitalWrite(MB_B, LOW);

    while(1)
    {
        if (stepsLeft >= steps)
        {
            analogWrite(MA_PWM, 0);
```

```

}
if (stepsRight >= steps)
{
    analogWrite(MB_PWM, 0);
}
if (stepsLeft >= steps && stepsRight >= steps)
{
    analogWrite(MA_PWM, 0);
    analogWrite(MB_PWM, 0);
    break;
}
}
}

void moveBackward(byte steps, unsigned int &stepsLeft, unsigned int &stepsRight)
{
    stepsLeft = 0;
    stepsRight = 0;
    analogWrite(MA_PWM, 100);
    digitalWrite(MA_F, LOW);
    digitalWrite(MA_B, HIGH);

    analogWrite(MB_PWM, 100);
    digitalWrite(MB_F, LOW);
    digitalWrite(MB_B, HIGH);

    while(1)
    {
        if (stepsLeft >= steps)
        {
            analogWrite(MA_PWM, 0);
        }
        if (stepsRight >= steps)
        {
            analogWrite(MB_PWM, 0);
        }
        if (stepsLeft >= steps && stepsRight >= steps)
        {
            analogWrite(MA_PWM, 0);
            analogWrite(MB_PWM, 0);
            break;
        }
    }
}

void rotateLeft(byte steps, unsigned int &stepsLeft, unsigned int &stepsRight)
{

```

```

stepsLeft = 0;
stepsRight = 0;
analogWrite(MA_PWM, 100);
digitalWrite(MA_F, HIGH);
digitalWrite(MA_B, LOW);

analogWrite(MB_PWM, 100);
digitalWrite(MB_F, LOW);
digitalWrite(MB_B, HIGH);

while(1)
{
    if (stepsLeft >= steps)
    {
        analogWrite(MA_PWM, 0);
    }
    if (stepsRight >= steps)
    {
        analogWrite(MB_PWM, 0);
    }
    if (stepsLeft >= steps && stepsRight >= steps)
    {
        analogWrite(MA_PWM, 0);
        analogWrite(MB_PWM, 0);
        break;
    }
}

void rotateRight(byte steps, unsigned int &stepsLeft, unsigned int &stepsRight)
{
    stepsLeft = 0;
    stepsRight = 0;
    analogWrite(MA_PWM, 100);
    digitalWrite(MA_F, LOW);
    digitalWrite(MA_B, HIGH);

    analogWrite(MB_PWM, 100);
    digitalWrite(MB_F, HIGH);
    digitalWrite(MB_B, LOW);

    while(1)
    {
        if (stepsLeft >= steps)
        {
            analogWrite(MA_PWM, 0);
        }
    }
}

```

```

if (stepsRight >= steps)
{
    analogWrite(MB_PWM, 0);
}
if (stepsLeft >= steps && stepsRight >= steps)
{
    analogWrite(MA_PWM, 0);
    analogWrite(MB_PWM, 0);
    break;
}
}
}

```

```

void moveManipulator(byte value)
{
    int oldPos = manipulator.read();
    if (value > oldPos)
    {
        for (int i = oldPos; i <= value; i++)
        {
            manipulator.write(i);
            delay(15);
        }
    }
    else
    {
        for (int i = oldPos; i >= value; i--)
        {
            manipulator.write(i);
            delay(15);
        }
    }
}

```

```

byte magnet_value = 0;

```

```

void magnetOn()
{
    for (int i = magnet_value; i < 255; i++)
    {
        analogWrite(MAGNET, i);
        delay(1);
    }
    magnet_value = 255;
}

```

```

void magnetOff()

```



```

{
  for (int i = magnet_value; i > 0; i--)
  {
    analogWrite(MAGNET, i);
    delay(1);
  }
  magnet_value = 0;
}

void setup() {
  Serial.begin(9600);
  bluetooth.begin(9600);
  manipulator.attach(6);
  manipulator.write(170);
  attachInterrupt(digitalPinToInterrupt(3), incrementStepsLeft, CHANGE);
  attachInterrupt(digitalPinToInterrupt(2), incrementStepsRight, CHANGE);
}

byte opcode = 0;
byte value = 0;

void loop() {
  if (bluetooth.available() >= 2)
  {
    opcode = bluetooth.read();
    value = bluetooth.read();

    switch (opcode)
    {
      case 16:
        moveBackward(value, stepsLeft, stepsRight);
        break;
      case 17:
        moveForward(value, stepsLeft, stepsRight);
        break;
      case 32:
        rotateLeft(value, stepsLeft, stepsRight);
        break;
      case 33:
        rotateRight(value, stepsLeft, stepsRight);
        break;
      case 48:
        moveManipulator(value);
        break;
      case 64:
        if (value == 0)
          magnetOff();
    }
  }
}

```

```
if (value == 255)
    magnetOn();
break;
}
bluetooth.write('!'); // done package
```