Моделирование одометрической системы автономной навигации гусеничного робота, оптимальной по критерию производительности

А.И. Данилко[1], *О.В. Стукач*[1,2]

[1]Национальный исследовательский университет "Высшая школа экономики", Москва, Россия
[2]Новосибирский государственный технический университет, Новосибирск, Россия

Аннотация: Моделируется система навигации автономного самоходного аппарата, передвигающегося по открытой местности. Рассматривается возможность использования одометрии в качестве навигационной системы для робота при выполнении задания и возвращения в исходное положение, приведена программная реализация. Дан обзор других решений обеспечения навигации роботов, определяется возможность их использования в качестве дополнительного инструмента для повышения точности определения координат. Реализован алгоритм преобразования географических координат в новую координатную плоскость, алгоритм поиска маршрута к цели, механизм вычисления текущей собственной позиции на основе совершенных перемещений с механизмом внесения правок погрешностей на основе данных, полученных от какого-либо периферийного устройства, например, гироскопа, алгоритм возврата к точке отправления. Приведено описание программных модулей, реализующих выбранные решения, оценена их эффективность.

Ключевые слова: Робототехника, программное управление, одометрия, определение координат, навигация робота

Введение

В 2020 году было выпущено более 2,7 миллионов промышленных роботов. Стремительное развитие робототехники привело к расширению спектра задач, решаемых роботизированными системами: от промышленных и бытовых до военных [1]. Одновременно с созданием передвигающихся роботов возник вопрос об определении собственного местоположения – навигации. Но существующие способы автономной навигации далеки от совершенства в части накопления ошибок в определении координат, что недопустимо для самоходных военных роботов без возможности постоянной проверки текущего местоположения. Эта область робототехники продолжает активное развитие и является крайне востребованной: в США серьезную часть военного бюджета составляет развитие военных роботов, как и в других странах, активно участвующих в военных конфликтах [2].

Для корректного выполнения боевой задачи роботу необходимо знать свое местоположение. В статье предлагается такой механизм навигации и связанный с ним механизмам передвижения, который позволит выполнить функцию артобстрела и возврата в исходную точку. Наиболее часто роботы применяются на ровной открытой местности, где потери личного состава могут быть наиболее велики, и роботы наилучшим образом могут помочь решить боевую задачу.

В статье рассмотрен принцип работы одометрии в других устройствах и проблемные ситуации, с которыми сталкивались при использовании этих систем. Исследованы другие методы решения задачи определения местоположения робота. После оценки их достоинств и недостатков были выбраны решения, которые могут нивелировать проблемы, встречающиеся в ходе использования одометрии, например решение, способное обеспечить объезд препятствий. Проведены расчёты и получена итоговая формула определения координат. Для успешного моделирования к каждой функции аппарата написан отдельный программный модуль, обеспечено их взаимодействие. Объединенные модули будет возможно перенести на конкретное устройство, заменив условные сигналы на конкретные команды для каждого элемента устройства, что обеспечит широкие возможности модернизации устройств.

1. Методы автономной навигации

Эффективность методов автономной навигации зависит от назначения робота, поэтому разделим обзор на разбор решений, используемых для избегания препятствий, без которых невозможна реализация автономного робота, и на обзор способов определения роботом своего местоположения на местности.

Мобильные роботы на гусеничном ходу имеют широкий спектр применений, включая поисково-спасательные работы в городах. В работе [3] изучается взаимосвязь между сложностью модели и производительностью. Обеспечен высокий уровень сходства физического взаимодействия гусеничного движителя с опорной поверхностью.

При проектировании автономного робота следует решить задачу объезда препятствий, так

как иначе передвижение объекта невозможно даже на известной территории. Примером этого могут служить автопилотируемые машины [4]. Самым привлекательным решением является воспроизведение механизмов мышления человека, которые используют как аналитическую составляющую, так и содержащуюся в памяти информацию. Подобную реализацию предложили авторы статьи [5]. Помимо сенсоров, отвечающих за проверку наличия препятствия перед собой, робот также использовал камеру, которая опознавала ближайшие объекты для выполнения периферийных функций, а также служила дополнительным детектором препятствий. Дополняла всё это внутренняя семантическая карта, которая выступала некоторым аналогом человеческой памяти [6]. Главным недостатком данного метода является необходимость длительного обучения в конкретной области, то есть для корректной работы такой стратегии необходимо заранее внести карту местности и препятствий, что неприменимо в заданных в работе условиях [7].

Ослабить данное ограничение попыталась группа исследователей [8], используя вместо карты, полученной на основании длительного обучения, наброски, где был нарисован предполагаемый маршрут передвижения робота. Результат можно считать успешным: даже при изменении расстояния между объектами (препятствиями) и перестановки их местами, робот успешно следовал схематическому курсу. Данный метод может быть использован для объезда крупных препятствий, например, зданий или же патрулей противника.

Становится ясно, что необходим механизм обнаружения мелких случайных препятствий. Эта задача системно рассмотрена в [9], где даны возможности как комплекса устройств для обнаружения объектов, так и каждого в отдельности. Наиболее часто для определения препятствий используют лидар – технологию получения и обработки информации с помощью активных оптических систем, использующих явление поглощения и рассеяния света в оптически прозрачных средах [10]. Подробно механизм работы устройства рассмотрен в статье [11]. При его внедрении выявлены основные недостатки лидарной технологии, такие как возможная потеря узких горизонтальных препятствий, например натянутой колючей проволоки, а также относительная дороговизна [12].

В исследовании [13] представлена адаптивная стратегия управления системами с обходом препятствий при наличии неисправностей привода и изменяющихся во времени неопределенностей. Стратегия основана на подходе "лидер-преследователь". Предполагая, что движение лидера задано. Чтобы избежать столкновений с препятствиями, был реализован метод центроидной тесселяции Вороного (*Centroidal Voronoi Tessellation, CVT*), а для реализации *CVT* сконструирован контроллер на основе аппроксимации функций и инвариантности движения.

Следующим способом является использование видеокамеры с обученной на распознавание препятствий нейросетью [14]. В последние несколько лет возможности нейросетевых технологий существенно возросли, в том числе в области компьютерного зрения [15–16]. Способность распознавать предметы на полученных кадрах и, как следствие, работать во время движения аппарата, может позволить выявить препятствия. В этом случае должно быть произведено предварительное обучение. Однако в этот раз нет необходимости обучения на той же области, где предполагается работа. Достаточно будет обучить нейросеть, взаимодействующую с выбранной камерой на местности с ландшафтом, схожим с предполагаемым местом действий. Следует отметить потребность в качественной камере и увеличение требований к ресурсам обработки информации для корректной работы нейросети [17–18].

В статье [16] представлены экспериментальные результаты локализации робота в помещении с использованием одной внешней камеры RGB-D. Были рассмотрены три типа движения робота в виртуальной офисной среде: статическое состояние, линейное движение и три различных случая криволинейного перемещения. Во всех случаях использование внешней камеры позволило получить достаточно точное местоположение робота.

Для условия ровного ландшафта кажется возможным применение ультразвукового эхолокатора [19]. Для предполагаемого устройства отсутствие отраженных звуковых волн будет означать, что в ближайшем окружении нет препятствий, однако снова следует заранее изучить фон, который будет получен при движении аппарата, то есть то, какой сигнал будет возвращаться в случае идеальных условий ровной поверхности с последующим вычитанием оного при поиске препятствий.

Последним из предлагаемых к использованию методов является радар – как и в случае с эхо локацией ультразвука, здесь используется принцип отражение радиоволн от предполагаемых объектов [20], но двигаться должен сам радар, а не искомые объекты. Также есть возможности комбинирования радара с нейросетями, настроенными на распознавание препятствий, как было предложено поступить для видео обнаружения. Подобное решение успешно реализовано в работе [21].

Итак, механизм объезда препятствий реализуется отдельно от навигации, а значит можно моделировать его в виде условного модуля, от которого проектируемой системе

навигации будут поступать сигналы о необходимости изменения маршрута. При моделировании целесообразно предположить, что опознавание препятствия производится нейросетью на основе полученной от датчика информации. Тогда при моделировании возможно заменить базу данных обученной нейросети на матрицу, содержащую координаты препятствий, в результате чего модуль будет обрабатывать попадающие в диапазон датчика координаты и сверять их с базой существующих препятствий.

2. Методы ориентирования на карте

Наиболее популярный метод ориентирования на карте – это *GPS* навигатор, который сейчас имеется в каждом современном телефоне [22]. Система глобального позиционирования определяет местоположения объекта путём измерения моментов времени приёма синхронизированного сигнала от навигационных спутников антенной потребителя. Результатом расчетов будут являться координаты объекта, представленные в виде долготы и широты, с точностью, зависящий от конкретного устройства и колеблющейся от десятков сантиметров до десятков метров [23]. Для предполагаемого робота предполагается массовое использование, то есть групповой обстрел территории большого радиуса, так что подобные погрешности допустимы [24]. Главным недостатком *GPS* является его известность, то есть противник, зная принципы работы системы, может полностью нарушить работу системы. Авторы статьи [25] подробно изучили возможность посылки ложных GPSсигналов и пришли к выводу, что таким образом можно полностью сбить систему навигации аппарата, а в данном случае боевого средства и, зная координаты цели, возможно даже его применение с помощью подмены сигналов. Также авторы пришли к выводу, что шифрование является обязательным средством защиты, но не гарантирует безопасности, поэтому следует учесть это при выборе способа навигации.

Также весьма привлекательными кажутся решения получать информацию от фиксированного источника в точке отправления, например, с помощью Интернета. При постепенном внедрении покрытия сетей пятого поколения этот способ кажется всё менее фантастическим [26]. К сожалению, возникает та же опасность, что и для *GPS* навигации: возможности перехвата данных, которые возможно либо заменить, либо накрыть фоном, в результате чего будут получены неточные данные [27]. Из этого можно сделать вывод, что данные системы не могу быть использованы как основное средство навигации военного робота.

Использование таких систем как вспомогательных (*Рис*. 1) было реализовано для глубинных исследовательских аппаратов [28]. Для навигации автономного аппарата основным средством навигация являлась одометрия, то есть определение своего местоположения на основе отданных команд передвижения [29]. Данная система ориентации в пространстве позволяет обеспечить полную навигационную автономию. В статье автономная навигация давала отклонение несколько десятков метров в час, то есть примерно такую же ошибку, которую выдают некоторые системы GPS. Здесь также стоит учесть куда более сложную систему аппарата, чем та, которую предполагаем мы, а также влияние течения [30], которое не учитывается системой. Компенсировать данные потери возможно введением дополнительной системы корректировок, например гироскопа [31–32].

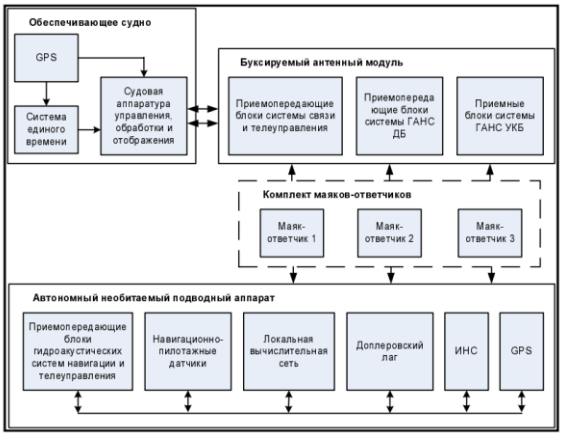


Рис. 1. Система навигации автономного глубинного аппарата

Таким образом, можно прийти к выводу о возможности реализации системы автономной навигации для предложенной модели робота: при начале выполнения задания ему будут передаваться текущие географические координаты и координаты цели, после чего, зная свое текущее местоположение, робот может использовать одометрию для достижения точки обстрела, не рискуя при этом получить ложные сигналы или потерять связь с базой, а после чего потеряться. Подобный пример рассмотрен в статье [33]. Во время движения навигация на основе отданных команд будет сверяться с показаниями гироскопа, тем самым контролируя накапливающуюся ошибку, а оптические системы способны реализовать объезд препятствий вне самой системы навигации.

Поскольку предполагается, что аппарат выполняет периферийную функцию артобстрела, то целесообразно оценивать результаты работы по качеству выполнения этой задачи. Тогда следует ввести расстояние стрельбы, оказавшись в пределах которого аппарат перейдет в режим выполнения

периферийной функции, то есть артиллерийской стрельбы. Из предполагаемой аппаратом точки местности производится наводка на координаты цели и следует выстрел по рассчитанным параметрам. В рассматриваемом случае же известно точное местоположение аппарата, в которое подставляются параметры, после чего моделируем сам выстрел. Механизм моделирования выстрела из артиллерийской установки был предложен авторами статьи [34]. Тогда, определив предполагаемую модель орудия, возможно подставить параметры используемого патрона в формулу баллистики снаряда и корректно смоделировать выстрел, результат которого изучить по предложенной методике [35], тем самым оценив эффективность проведенной работы.

3. Разработка алгоритмов моделирования

Определившись с моделируемым устройством и методами навигации, можно приступить к проектированию структуры программы. Первое, что необходимо сделать – это преобразовать географические координаты в собственную координатную плоскость. Для этого необходимо также рассчитать расстояние между данными координатами [36].

Построив свою координатную плоскость, необходимо создать на ней препятствия для реализации механизмов объезда. Так как основная работа будет проходить с использованием внутренних координат, то возможно создать матрицу, которая будет содержать все координаты, в которых имеются препятствия.

При проведении анализа литературы, в качестве способа обнаружения препятствий был выбран механизм анализа видеопотока, получаемого с камеры, обученной нейросетью. Модуль, имитирующий работу камеры на устройстве, должен перекрыть некоторую зону видимости. Для этого возможно записывать в обновляющуюся матрицу координаты, полученные на основе текущего местоположения. Тогда база координат препятствий может заменить базу знаний нейросети и сравниваться с матрицей координат, попадающих в зону видимости. В случае совпадения будет генерироваться сигнал об обнаружении препятствия. Кроме того, необходимо реализовать объезд препятствий. Основная суть данного модуля состоит в перехвате контроля у алгоритма поиска пути. То есть, по сути, он должен делать тоже самое, только искать путь к точке начальной прямой, но за препятствием. Так как данный алгоритм нужен лишь для полноты, то можно ограничиться его достаточностью для преодоления сгенерированных препятствий.

Далее следует создать систему поиска пути, которая будет управлять передвижением. На входе она должна получить координаты из системы навигации, на основе которых отдавать команды передвижения.

На этапе создания программы до её формализации и условного усложнения командам передвижения достаточно изменять угол поворота устройства в системе координат и скорость передвижения. На основе этих двух параметров происходит изменение как реальных координат с дополнительными параметрами в виде коэффициента трения [37], наличия ям, камней, неровностей и т.д., так и навигационных, для которых предполагается наличие системы корректировок (гироскопов) [38].

Следующим шагом является создание модуля, имитирующего артиллерийский обстрел. При минометном обстреле снаряд при детонации получает стартовую скорость и далее летит под выбранным углом к горизонту, поэтому для моделирования целесообразно использовать соответствующие формулы, добавив случайную ошибку к параметрам наведения и скорости снаряда.

Последним этапом является визуализация результатов. Для этого достаточно в дискретные моменты времени записывать текущие координаты в отдельный массив, который потом вывести на график. Для визуализации стрельбы достаточно продемонстрировать места попаданий, очертив вокруг них зону поражения, а также радиус цели. Схема взаимодействия модулей программы представлена на *Рис. 2*.

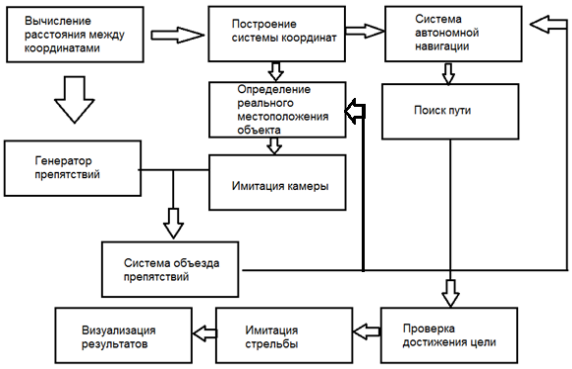


Рис. 2. Диаграмма программного обеспечения

4. Преобразование географических координат

Для нахождения расстояния между двумя точками на сфере необходимо для начала рассчитать угловую разницу между ними, а затем умножить её на радиус сферы (*Рис.* 3).

Рис. 3. Нахождение длины дуги

Наиболее подходящей формулой для нахождения угловой разницы является формула гаверсинусов, так как она лучше работает на малых расстояниях:

$$\Delta\sigma = 2\arcsin[\sin^2(\frac{\varphi_2 - \varphi_1}{2}) + \\ + \cos\varphi_1 \cos\varphi_2 \sin^2(\frac{\Delta\lambda}{2})], \quad (1)$$

где φ, λ – соответственно географическая широта и долгота двух точек 1, 2 в радианах.

Таким образом, необходимо преобразовать разницу географических координат в радианы и подставить в предложенную формулу (1), результат которой умножить на радиус сферы, в нашем случае радиус Земли.

На *Рис. 4* представлен код модуля, выполняющего данные расчеты: на входе задаются географические координаты, результат записывается в переменную *dist*.

```
rad = 6372795
llatl = 33.010200
llongl = 36.021280
llat2 = 33.510200
llong2 = 36.291280
latl = llatl * math.pi / 180.
lat2 = llat2 * math.pi / 180.
longl = llongl * math.pi / 180.
long2 = llong2 * math.pi / 180.
cll = math.cos(latl)
cl2 = math.cos(lat2)
sll = math.sin(latl)
sl2 = math.sin(lat2)
delta = long2 - longl
cdelta = math.cos(delta)
sdelta = math.sin(delta)
y = math.sqrt(math.pow(cl2 * sdelta, 2) +
math.pow(el1*sl2-sl1*cl2*cdelta, 2))
x = sl1*sl2 + cll *cl2 *cdelta
ad = math.atan2(y, x)
dist = ad * rad
```

Рис. 4. Модуль расчета расстояния

5. Создание набора препятствий

Как было определено при разработке структуры программы, необходимо записать в некий общий список координаты всех точек, где есть препятствия. Чтобы продемонстрировать механизм объезда препятствий, они должны повстречаться на пути устройства, а значит необходимо расположить их на прямой между целью и объектом.

Во избежание использования отрицательных координат при построении новой координатной плоскости система навигации устанавливает начальное значение устройства по *x* на *x* = *dist*/2, тогда координаты цели будут *x* = *dist*/2, *y* = *dist*. Тогда препятствия необходимо генерировать на прямой между этими координатами. На *Рис. 5* представлено создание трёх «кирпичей» шириной в 400 метров и длиной в 250, центры которых лежат на пути устройства.

Далее происходит запись в список координат точек, входящих в препятствие, который после используется камерой как база данных. На данном этапе решения читаемость кода важнее быстродействия, а сравнение двух списков реализуется проще и порождает меньше возможных ошибок, чем очерчивание границ и сравнение их пересечений, поэтому было решено использовать данный метод для обнаружения препятствий.

6. Имитация работы камеры

Модуль, имитирующий работу камеры, должен соответствовать генератору препятствий в виде обрабатываемых данных, то есть работать с теми же объектами. Тогда наиболее очевидным решением является записать точки на расстоянии видимости перед объектом в список и сравнить его со списком препятствий, после чего определить, с какой стороны находится препятствие и передать эти данные в систему объезда препятствий.

```
for i in range(400):
xe = 30300 + c
xel = 30300 + c
xe2 = 30300 + c
c = c+1
y=0
for j in range(250):
ye = 20000 + v
yel = 40000 + v
ye2 = 55000 + v
v = v+1
di.append(xe)
yi.append(ye)
di.append(xe1)
yi.append(ye1)
di.append(xe2)
yi.append(ye2)
nestonks.add((xe2, ye2))
nestonks.add((xe1, ye1))
nestonks.add((xe, ye))
```

Рис. 5. Генератор препятствий

Для упрощения кода данный модуль генерирует три блока (*Рис.* 6), создавая фигуру (см. *Рис*. 7).

```
for i in range(20):
xes = xbasa - 10+c
c=c+1
v =  0
for j in range(20):
yes = ybasa+v
v =v+1
real_xes=round(math.cos(hangle2) * xes+math.sin(-hangle2) * yes +xbasa2)
real_yes = round(math.sin(hangle2)*xes+math.cos(hangle2)* yes+ybasa2)
nestonks2.append((real_xes,  real_yes))
```

Рис. 6. Создание части обзора

Рис. 7. Зона видимости камеры

Также необходимо учесть, что при повороте камеры изменится угол обзора, а значит в поле зрения попадут другие координаты. Для этого необходимо выполнить матричное преобразование (*Рис*. 8):

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} X_p \\ Y_p \end{bmatrix},$$

которое, в зависимости от угла пересчитает внутренние координаты, началом координатной плоскости которых является (0, 0) и отсчёт ведется от самого объекта. Повернув систему координат, программа добавляет к ним реальные координаты, и получается набор точек для другого угла обзора (*Рис.* 9).

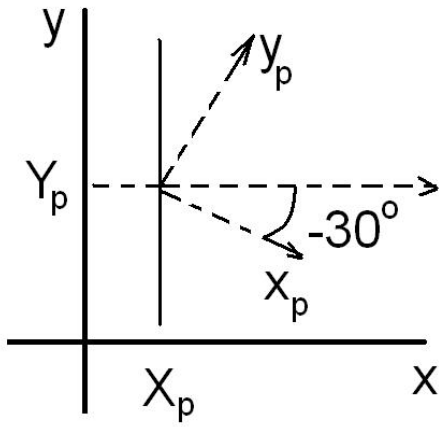


Рис. 8. Преобразование координат

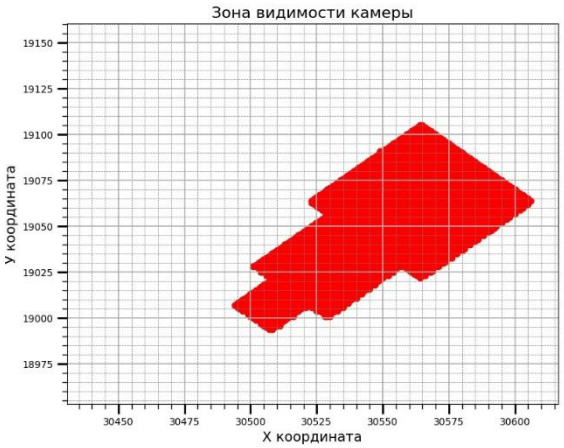


Рис. 9. Зона видимости при повороте на 45 градусов

8. Объезд препятствий

Объезд препятствий вынесен в отдельную функцию, которая вызывается при каждой итерации цикла, но запускается только при поступлении разрешающего сигнала (*Рис. 10*). Это обеспечивает перехват системы поиска пути, так как она не запустится, пока не закончится объезд препятствия.

```
Cam(real_obj_coor_X, real_obj_coor_Y)
ifosQ
exai(bi5)
```

Рис. 10. Порядок вызова функций

В начале цикла вызывается модуль камеры, который формирует список видимых точек, которые после сравниваются со списком препятствий функцией ifos(), и при их обнаружении, в зависимости того, с какой стороны от самого устройства находится ближайшее препятствие, передает разрешающий сигнал и направление объезда. Функция предполагает прямоугольный объезд (четыре

поворота на 90 градусов) и возвращение на начальную траекторию с проверкой при начале очередного поворота на препятствие перед собой. Результат работы представлен на *Рис. 11*.

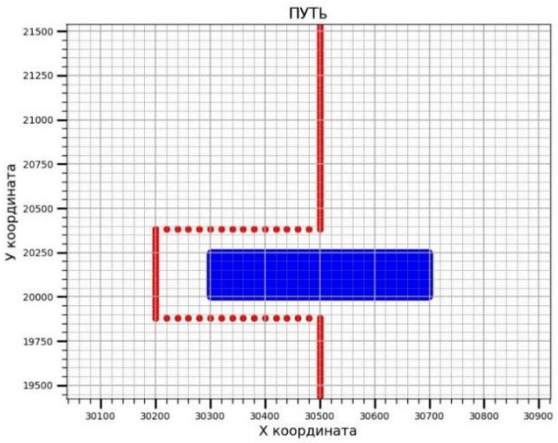


Рис. 11. Траектория объезда препятствия

9. Расчет местоположения объекта и система навигации

Очевидно, что движение по поверхности не может быть идеальным и содержит некоторые потери, точность которых невозможно учесть в самой навигации, так как они неизвестны заранее. Для расчёта реальных координат была создан код с набором случайных переменных. В основе кода лежит текущая скорость устройства и его угол в системе координат (*Рис. 12*). Случайные события изменяют как угол, так и вклад скорости в изменение положения тела.

```
if dt = = 30:
dt =0
kamushki = random.randint(-5, 5)
ym = random.randint(-5, 5)
znak = random.randint(0, 1)
rgrs = random.randint(997, 1003)
rgry = random. randint(997, 1003)
if znak = = 0:
znak = - 1
real_obj_coor_X = real_obj_coor_X+speed * math.cos(tangle) * (1 - kamushki / 1000) * (1 - ym / 1000)
real_obj_coor_Y = real_obj_coor_Y+speed * math.sin(tangle) * (1 - kamushki / 1000) * (1 - ym / 1000)
tangle = tangle + tangle * znak * (1-kamushki / 10000) * (1 - ym / 10000) / 100
```

Рис. 12. Случайные ошибки при движении

В это же время система навигации использует для определения местоположения только скорость, которую устанавливает устройству и предполагаемый угол к цели. Один раз в некоторое количество времени система навигации обращается к системам дополнительной корректировки – гироскопам (*Рис. 13*), которые фиксируют реальные изменения положения тела с некоторой точностью. Сравнивая эти изменения с предполагаемыми изменениями за тот же промежуток времени, система навигации находит ошибку и вносит корректировки, сокращая накопленную ошибку.

```
drX = real_obj_coor_X - newX
drY = real_obj_coor_V - newY
drT = newT - tangle
drXl = navX - newXl
drY1= navY - newYl
drTl = newTl - tangle
gyroSP   = (drX - drXl) * rgrs / 1000
gyroSPY = (drY - drYl) * rgrs / I0000
gyroHY   = (drT - drTl) * rgry / 1000
hangle   = hangle+gyroHY
navX =  navX+math.cos(hangle)*gyroSP
navY =  navY+irath.sin(hangle)*gyroSPY
```

Рис. 13. Системы корректировки

10. Поиск пути

На каждом шаге цикла вызывается функция проверки на достижение цели и, как только расстояние до объекта будет меньше определённого, например, одного километра, произойдет остановка, отладка угла наведения и начало стрельбы. До тех пор, пока цель не достигнута, с определённой периодичностью происходит перестроение маршрута (*Рис. 14*). На основе координат в системе навигации считается угол, под которым необходимо двигаться, чтобы достичь цели по прямой. Если он не равен текущему углу, то вызывается изменение угла. На этапе отладки можно изменять его прямо, но при дальнейшем усложнении целесообразно использовать функции управления координат, которые потребуют сначала остановки объекта, а затем его поворота, что также вызовет некоторые потери точности и создаст ошибку, тем самым периодичность вызова системы поиска пути будет более обоснована.

```
dox = target_coor_X - navX
doy - target_coor_V - navY
EL = math.hypot(dox, doy)
ttangle = math.acos(dox / EL)
if doy > 300 or math.hypot(dox, doy) > 400:
if dt6 = = 40:
dt6 = 0
hangle = ttangle
tangle = hangle
else: dt6 = dt6 + 1
else:
   break
```

Рис. 14. Перестроение маршрута

11. Обстрел территории

По достижении точки, удовлетворяющей условию удалённости, происходит остановка устройства и начало стрельбы. Для наведения орудия используется известная формула полёта тела под углом к горизонту (*Рис. 15*). Начальная

скорость снаряда зависит от его модели, в данном случае она равна 211 м/с. После наведения угла на основе предполагаемого местоположения производится выстрел из реального местоположения с добавлением случайных составляющих к углу наведения, скорости снаряда и углу выстрела к горизонту.

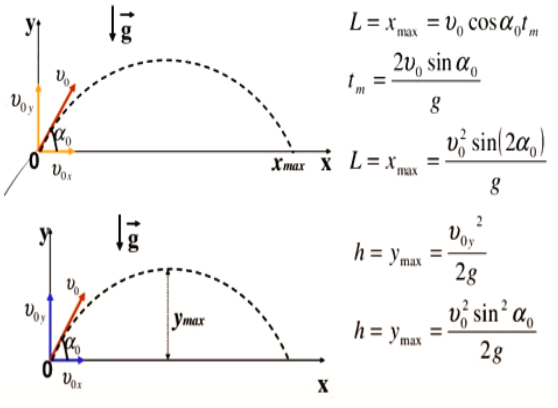


Рис. 15. Движение тела, брошенного под углом к горизонту

Для получения статистической оценки производится 10 выстрелов, места попаданий которых записываются в отдельный массив для последующей визуализации.

12. Визуализация результатов

В ходе работы программы на каждой итерации движения реальные и навигационные координаты записывались в списки (*Рис. 16*), как и места попаданий снарядов в соответствующем модуле. Имея наборы координат x и y, можно построить графики на основе библиотеки python – matplotlib.

```
xinl.append(real_oDj_coor_X)
yinl.append(real_oDj_coor_Y)
xin2.append(navX)
yin2.append(navY)
```

Рис. 16. Запись координат

На *Рис*. 17 и 18 можно видеть траекторию движения аппарата, где синим цветом отмечены точки, записанные системой навигации, а красным – реальные. Зелёные блоки – это препятствия, которые успешно объезжаются. На *Рис*. 19 показаны результаты стрельбы, где синим очерчена зона, попадание в которую считается успехом, а красный круг возле точек – зоны поражения снарядов.

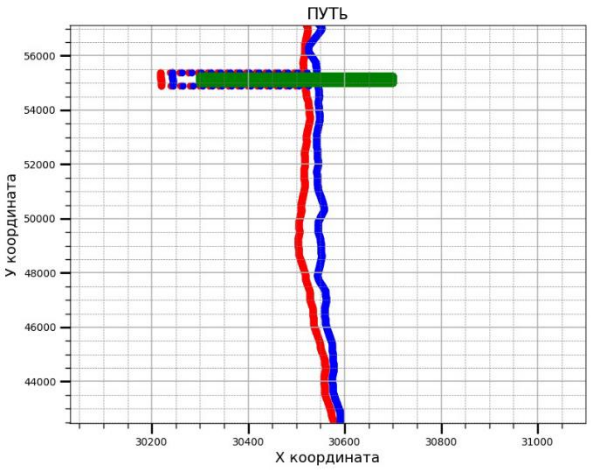


Рис. 17. Объезд препятствия

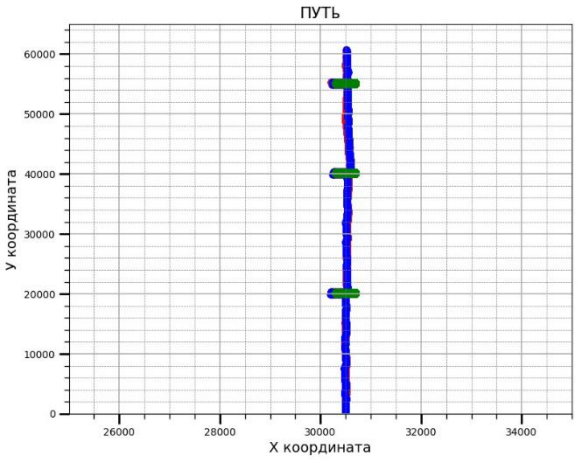


Рис. 18. Траектории движения

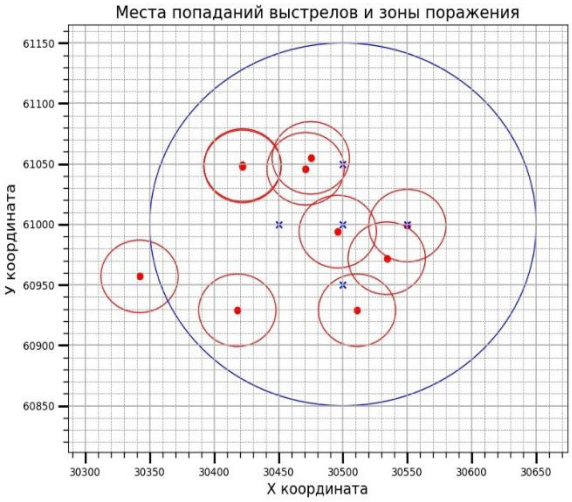


Рис. 19. Результаты стрельбы

13. Моделирование и оценка результатов

Для корректной оценки результатов работы программы и возможностей практического создания смоделированного устройства

необходимо правильно подобрать параметры, чтобы моделируемые условия были близки к реальным. Чтобы создать потери скорости и отклонения от траектории движения в программе использовался коэффициент трения поверхности, а также условные события в виде неровностей, которые с определённой частотой меняли скорость движения и угол поворота аппарата для правильного подбора значений этих параметров, а именно процентного уменьшения скорости и величины отклонения. При изучении современных систем контроля скорости, например, круиз-контроля в автомобилях, очевидно, точная оценка скорости путём сравнения данных с разных датчиков, например, дополнительного устройства на оси вращения колес со спидометром полностью решает проблему несоответствия реальной скорости и предполагаемой. Кроме того, данные системы способны отслеживать угловое отклонение и повороты колес, тем самым возвращая их на место, что позволит избежать резких изменений и критических несоответствий [39], на основе чего угловую ошибку можно свести к минимуму и использовать для демонстрации системы корректировок. В то же время отличие скорости транспортного средства на асфальтированной дороге и бездорожье может достигать почти 50 % при обилии ям, бугров или грязи, так как в рассматриваемых условиях предполагается пустынная ровная. Потери скорости могут достигать 20 % [40].

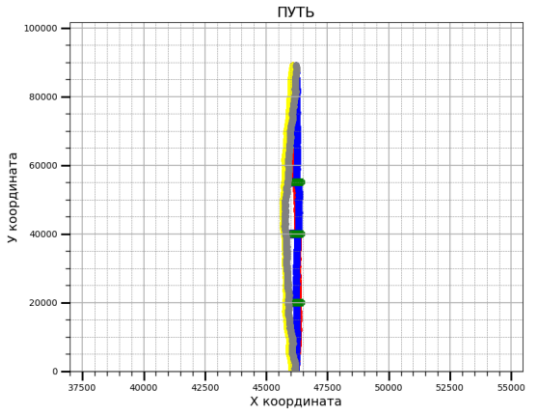
Следующим важным фактором для корректной оценки является точность системы корректировок, а именно гироскопа. Современные гироскопические системы рассчитаны на длительную работу в течение дней и месяцев, после которых начинает накапливаться значимая ошибка для моделируемого устройства, а время в пути до цели не превышает нескольких часов. Поэтому, опираясь на современные параметры гироскопов [41], можно определить ошибку в 1–3 % с перекосом в сторону большего значения, тогда случайный множитель целесообразно установить, как случайное значение в диапазоне (0,99; 1,02).

Заключительным элементом моделирования является стрельба. Здесь ошибки добавлены в начальную скорость снаряда, равную по умолчанию 211 м/с, углу к горизонту при калибровке выстрела и точности наведения, так как эти параметры являются частями формулы для определения точки попадания снаряда. Колебания угла горизонтальной наводки составляют ±3 градуса даже в довольно устаревших моделях орудий, поэтому случайную ошибку более чем достаточно установить в данных пределах. Так как скорость полёта снаряда зависит от типа снаряда и длины ствола орудия, то колебания начальной скорости практически отсутствуют и не указываются в параметрах орудия. Поэтому можно сделать данный элемент фиксированной величиной или сделать отклонение номинальным в несколько м/с. Последней изменяемой компонентой в расчётах является угол полета снаряда к горизонту, погрешность которого зависит от системы наведения, которая будет настраивать пусковую установку, что для современных высокоточных систем также является довольно простой задачей. Но можно учесть возможную неровность поверхности или же потери, вызванные отклонением снаряда ветром и трением воздуха, добавив их в случайную ошибку угла к горизонту вместо усложнения формулы. Так как сами системы наведения не создают ошибку, то она также фигурирует в крайне низких пределах и её можно оставить на уровне 1–3 градусов.

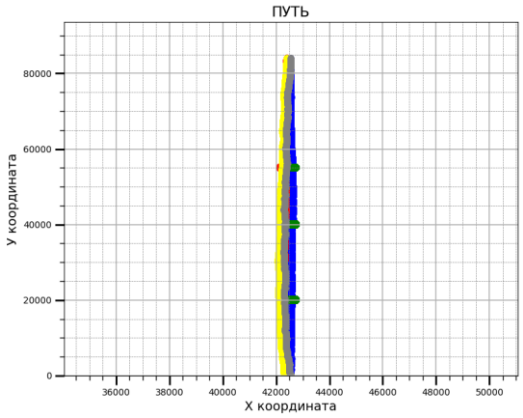
После подстановки в программу параметров, наиболее приближающих моделирование к реальным условиям, можно оценить комплексный результат работы программы. На *Рис*. 20 представлена полная траектория движения самоходного аппарата к заданной географическими координатами цели и возврата в точку отправления: красной линией отмечена реальная траектория движения к цели, а желтой – реальная траектория движения от цели. Синей линией представлена траектория, которая находилась в системе навигации на пути к цели, и на основе которой производился поиск пути; серая линия – данные системы навигации на обратном пути. На *Рис*. 21 данные приближены в точке, из которой производилась стрельба. На *Рис.* 22 красными точками отображены места попаданий снарядов с радиусом поражения. Радиус цели очерчен синим, где крестиками отмечены точки, по которым производилась стрельба.

Итоги моделирования, представленные на рисунках, показывают, что отклонение от предполагаемого курса не является критическим, что видно из разницы координат, а также успешного поражения цели. Накопленная ошибка не является серьёзной даже после возвращения к точке отправления, то есть система корректировки полностью нивелирует недостатки основного метода расчёта.

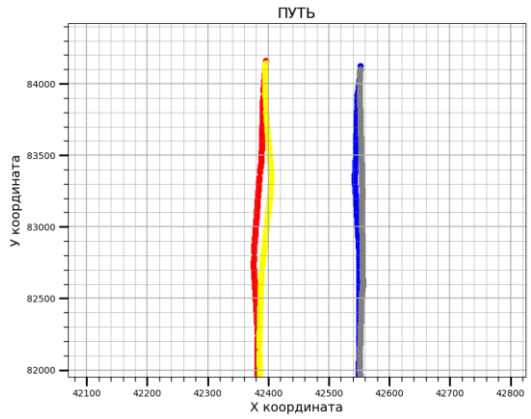
Оценивая результаты стрельбы, стоит также учесть, что выбранный размер цели в 250 метров является условностью, призванной увеличить наглядность проекта, в то время как целью могут быть куда более масштабные объекты, с радиусом в несколько километров, для которых полученная ошибка будет отсутствовать. Так как артиллерийский обстрел служит дополнительной функцией программы, призванной обеспечить наглядность, то для упрощения анализа результатов достаточно оценить поражённую обстрелом территорию.



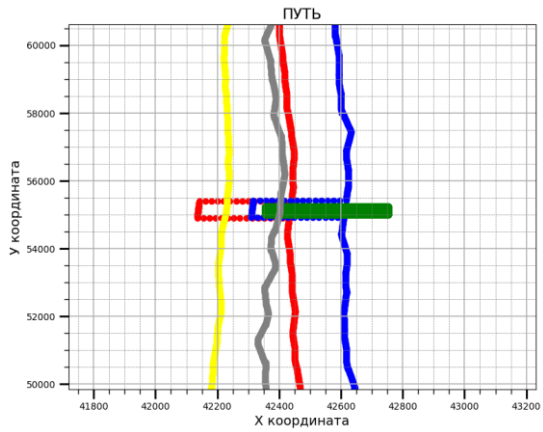
а)



б)

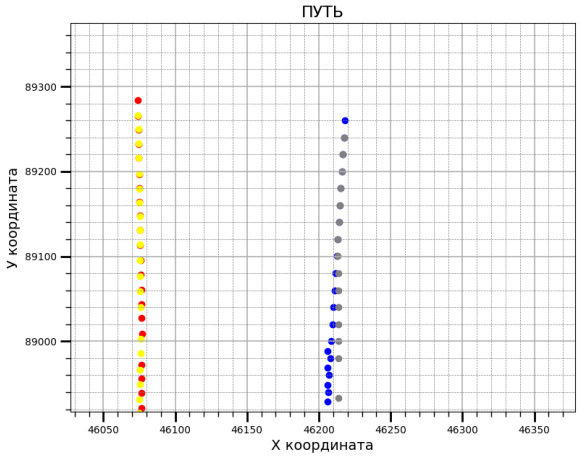


в)

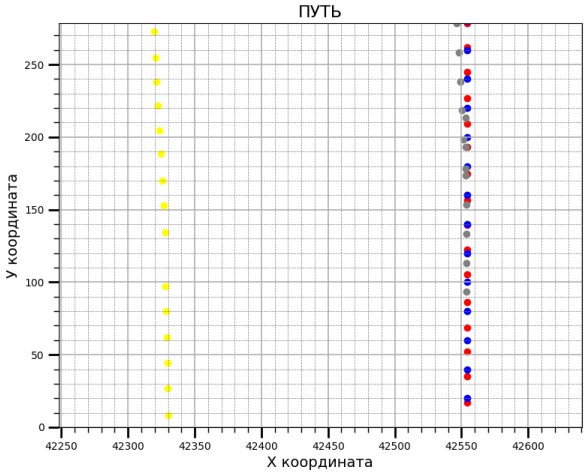


г)

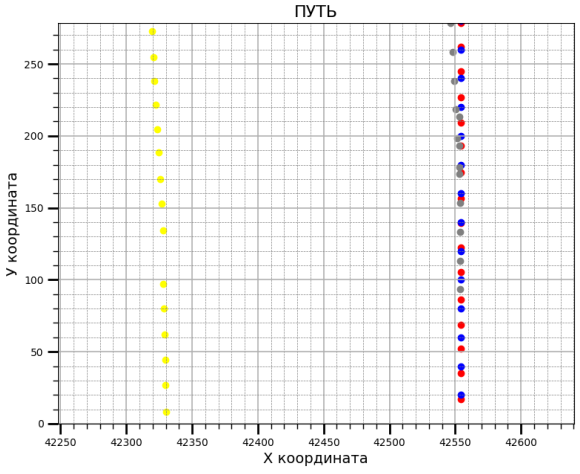
Рис. 20. Траектория движения самоходного аппарата к цели и обратно для разных координат



а)

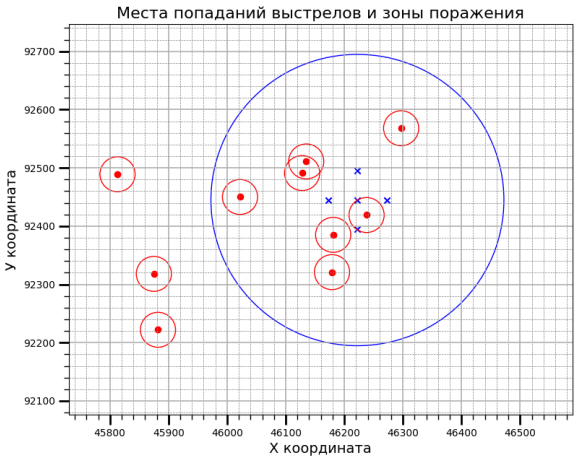


б)

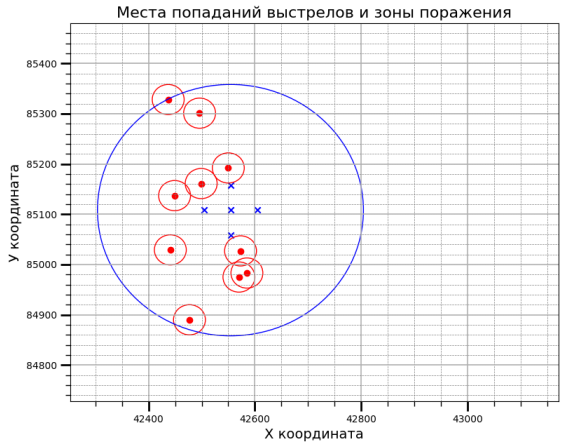


в)

Рис. 21. Координаты объекта при достижении цели движения для разных координат



а)



б)

Рис. 22. Результаты стрельбы по цели в заданных координатах

За 15 запусков моделирования с одинаковыми параметрами в среднем происходило 6,8 попаданий внутрь очерченного круга, каждое из которых имеет поражающий радиус 30 метров, что можно считать успешным выполнением боевой задачи, особенно для данного вида артиллерии.

Заключение

Несмотря на ряд параметров, призванных добавить случайную ошибку или же намерено понижающих точность расчётов, моделирование всё еще далеко от реальности, так как на практике параметров гораздо больше, и их значения меняются в зависимости от условий. Например, точность гироскопа зависит от его модели, а потери скорости зависят от поверхности. Кроме того, выстрел вызывает откат устройства, что также потребует дополнительных корректировок. Таким образом эффективность созданного аппарата будет в немалой степени зависеть от правильного взаимодействия элементов устройства и потребует дополнительной калибровки и отладки, прежде чем реальные результаты совпадут с моделированием.

Точная локализация важна для всех мобильных роботов, в которых приходится иметь дело с накапливающимися ошибками одометрии, бортовым сенсорным шумом, суровыми условиями окружающей среды, нестабильным или отсутствующим сигналом GPS, а также отсутствием или неопределённостью карты местности [12]. Для самоходного автономного артиллерийского аппарата были изучены и определены наиболее оптимальные методы обеспечения автономной навигации, исключающие риск перехвата контроля и сводящие к минимуму возможные ошибки в ходе выполнения поставленной задачи и возврата в точку отправления. Для предложенных методик были разработаны оптимальные для наглядности алгоритмы реализации, а также разработана система взаимодействия всех частей устройства. Также было произведено программное моделирование, которое продемонстрировало успешное взаимодействие спроектированной системы и корректную работу её частей.

Разработанная модель может послужить основой для реальной программы контроля автономного робота. Условные сигналы, которые не привязаны к конкретному техническому средству, но при этом однозначно понимаемые другими модулями программы, с легкостью заменяются на реальные команды передвижения, а моделируемые значения для имитирующих камеру датчиков заменены на получаемые сканером данные. Разделенная на множество модулей программа упростит отладку отдельных частей, а система визуализации позволит тестировать переписанные модули до их технической реализации, что сократит расходы при физической реализации проекта.

Результаты моделирования позволяют с уверенностью сказать, что создание и успешное функционирование самоходного аппарата с рассмотренной в работе системой автономной навигации имеет потенциал успешной реализации, что открывает коммерческие возможности для дальнейшей работы в изучаемой области и перспективы практического применения результатов дипломной работы. Данная работа может послужить в качестве основы для проекта по созданию автономных военных роботов и использоваться в качестве аргументированного предложения по запуску соответствующей разработки, в то же время имея в себе готовую архитектуру управления устройством.

Литература

[1] IFR presents World Robotics Report 2020 – International Federation of Robotics [Electronic resource]. URL: https://ifr.org/ifr-press-releases/news/record-2.7-million-robots-work-in-factories-around-the-globe?utm_source=CleverReach&utm_medium=ema

il&utm_campaign=NL+3%2F20&utm_content=Mailing_12155770 (врмя доступа: 07.03.2021).
[2] Анализ опыта боевого применения групп беспилотных летательных аппаратов для поражения зенитно-ракетных комплексов системы противовоздушной обороны в военных конфликтах в Сирии, в Ливии и в Нагорном Карабахе [Электронный ресурс]. URL: https://www.elibrary.ru/item.asp?id=44437305 (время доступа: 07.03.2021).
[3] Moskvin I., Lavrenov R., Magid E., Svinin M. Modelling a Crawler Robot Using Wheels as Pseudo-Tracks: Model Complexity vs Performance // IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA). – 2020. – Pp. 1–5. – Doi: 10.1109/ICIEA49774.2020.9102110. – https://ieeexplore.ieee.org/document/9102110.
[4] Ingle S., Phute M. Tesla Autopilot: Semi Autonomous Driving, an Uptick for Future Autonomy // Int. Res. J. Eng. Technol. – 2016. – Vol. 3. – № 9.
[5] Yan W., Weber C., Wermter S. Learning indoor robot navigation using visual and sensorimotor map information // Front. Neurorobot. – 2013. – Vol. 7. – № OCT.
[6] Ruiz-Sarmiento J.R., Galindo C., Gonzalez-Jimenez J. Building Multiversal Semantic Maps for Mobile Robot Operation // Knowledge-Based Syst. – 2017. – Vol. 119.
[7] Pattern Recognition and Machine Learning // J. Electron. Imaging. – 2007. – Vol. 16. – № 4.
[8] Li X. et al. A visual navigation method of mobile robot using a sketched semantic map // Int. J. Adv. Robot. Syst. 2012. – Vol. 9.
[9] Yu X., Marinov M. A study on recent developments and issues with obstacle detection systems for automated vehicles // Sustainability (Switzerland). – 2020. – Vol. 12. – № 8.
[10] Разработка системы автономной навигации на основе технологий лидарного сканирования местности и software defined radio // Международный научно-исследовательский журнал. – 2017. – № 7–3 (61).
[11] Wang D., Watkins C., Xie H. MEMS mirrors for LiDAR: A review // Micromachines. – 2020. – Vol. 11. – № 5.
[12] Debeunne C., Vivet D. A review of visual-lidar fusion based simultaneous localization and mapping // Sensors (Switzerland). – 2020. – Vol. 20. – № 7.
[13] Bai Y., Wang Y., Svinin M., Magid E., Sun R. Adaptive Multi-Agent Coverage Control With Obstacle Avoidance // IEEE Control Systems Letters. – 2022. – Vol. 6. – Pp. 944–949. – Doi:10.1109/LCSYS.2021.3087609. – https://ieeexplore.ieee.org/document/9448334.
[14] Zhan W. et al. Autonomous visual perception for unmanned surface vehicle navigation in an unknown environment // Sensors (Switzerland). – 2019. – Vol. 19. – № 10.
[15] Gu J. et al. Recent advances in convolutional neural networks // Pattern Recognit. – 2018. – Vol. 77.
[16] Kononov K., Lavrenov R., Tsoy T., Martiez-Garc E.A., Magid E. Virtual Experiments on Mobile Robot Localization with External Smart RGB-D Camera Using ROS. – IEEE 3rd Eurasia Conference on IOT, Communication and Engineering (ECICE). – 2021. – Pp. 656–659. – Doi: 10.1109/ECICE52819.2021.9645644. – https://ieeexplore.ieee.org/document/9645644.
[17] Roth W. et al. Resource-efficient neural networks for embedded systems // arXiv. 2020.
[18] Shukla D., Jha R.K. Robust motion estimation for night-shooting videos using dual-accumulated constraint warping // J. Vis. Commun. Image Represent. – 2016. – Vol. 38.
[19] Genzel D., Yovel Y., Yartsev M.M. Neuroethology of bat navigation // Current Biology. – 2018. – Vol. 28. – № 17.
[20] Crockett A., Rueckner W. Visualizing sound waves with schlieren optics // Am. J. Phys. – 2018. – Vol. 86. – № 11.
[21] Sligar A.P. Machine learning-based radar perception for autonomous vehicles using full physics simulation // IEEE Access. – 2020. – Vol. 8.
[22] Milner G. What is GPS? // J. Technol. Hum. Serv. – 2016. – Vol. – 34. – № 1.
[23] Ясюкевич Ю.В. и др. Влияние дифференциальных кодовых задержек GPS/ГЛОНАСС на точность определения абсолютного полного электронного содержания ионосферы // Геомагнетизм и аэрономия. – 2015. – Vol. 55. – № 6.
[24] Cai H. et al. Optimization of Maintenance Force Dispatch of Artillery Command Information System Based on Mission Success // Smart Innovation, Systems and Technologies. – 2021. – Vol. 190.
[25] Sorokin I., Obukhov A., Romanov et al. Application of a permutation decoding method in a small-class unmanned aerial vehicle control system (drones, multicopters) // Mod. Technol. Syst. Anal. Model. – 2019. – Vol. 2(62). – № 2(62). – P. 186–195.
[26] Kutuzov D., Osovsky A., Stukach O., Starov D. Modeling of IIoT Traffic Processing by Intra-Chip NoC Routers of 5G/6G Networks". International Siberian Conference on Control and Communications (SIBCON). 13–15 May 2021, Kazan, Russia. – DOI: 10.1109/SIBCON50419.2021.9438874. – https://ieeexplore.ieee.org/document/9438874.
[27] Zou Y. et al. A Survey on Wireless Security: Technical Challenges, Recent Advances, and Future Trends // Proceedings of the IEEE. –2016. – Vol. 104. – № 9.
[28] Manzanilla A. et al. Autonomous navigation for unmanned underwater vehicles: Real-time experiments using computer vision // IEEE Robot. Autom. Lett. – 2019. – Vol. 4. – № 2.
[29] Zhan H. et al. Visual Odometry Revisited: What Should Be Learnt? // IEEE International Conference on Robotics and Automation. – 2020.
[30] Steinberg J.M., Pelland N.A., Eriksen C.C. Observed evolution of a California Undercurrent eddy // J. Phys. Oceanogr. – 2019. – Vol. 49. – № 3.
[31] Tian Q., Wang K.I.K., Salcic Z. An INS and UWB fusion-based gyroscope drift correction approach for indoor Pedestrian tracking // Sensors (Switzerland). – 2020. – Vol. 20. – № 16.
[32] Маринушкин П.С., Бахтина В.А., Подшивалов И.А., Стукач О.В. Вопросы разработки инерциальных пешеходных навигационных систем на основе МЭМС-датчиков // Наука и образование. Электронное издание. ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". – N 06. - июнь 2015. – С. 157–173. – DOI: 10.7463/0615.0786740. – http://technomag.bmstu.ru/doc/786740.html.
[33] Yuan M., Yau W.Y., Li Z. Lost Robot Self-Recovery via Exploration Using Hybrid Topological-Metric Maps // IEEE Region 10 Annual International Conference, Proceedings/TENCON. – 2019. – Vol. 2018 – October.
[34] Rusyak I.G., Tenenev V. Modeling of ballistics of an artillery shot taking into account the spatial distribution of parameters and backpressure // Comput. Res. Model. – 2020. – Vol. 12. – № 5.

[35] Богатинов С.В. Эффективность выполнения огневой задачи – критерий оценки // Известия Российской академии ракетных и артиллерийских наук. – 2008. – № 1 (57). – P. 38–40.
[36] Bentsen M. et al. Coordinate transformation on a sphere using conformal mapping // Mon. Weather Rev. – 1999. – Vol. 127. – № 12.
[37] Llavori I. et al. Critical Analysis of Coefficient of Friction Derivation Methods for Fretting under Gross Slip Regime // Tribol. Int. – 2020. – Vol. 143.
[38] Park S. Adaptive control of a vibratory angle measuring gyroscope // Sensors. – 2010. – Vol. 10. – № 9.
[39] Fu H., Yu R. LIDAR scan matching in off-road environments // Robotics. – 2020. – Vol. 9. – № 2.
[40] Yu B. et al. Optimization and testing of suspension system of electric mini off-road vehicles // Sci. Prog. – 2020. – Vol. 103. – № 1.
[41] Мусалимов В.М., Ротц Ю.А., Астафьев С.А., Амвросьева А.В. Расчёт надёжности упругих элементов микромеханических гироскопов [Электронный ресурс]. – URL: https://www.studmed.ru/musalimov-v-m-rotc-yu-a-astafev-s-a-amvroseva-a-v-raschet-nadezhnosti-uprugih-elementov-mikromehanicheskih-giroskopov_a819c2f85c4.html (время доступа: 06.05.2021).

Приложение
Программа моделирования робота

```
import numpy as np
import math
import random
import matplotlib.pyplot as pit
from matplotlib.ticker import (MultipleLocator,
FormatStrFormatter, AutoMinorLocator)
rad = 6372795
llat1 = 33.010200
llong1 = 36.021280
llat2 = 33.710200
Ilong2 = 36.391280
lat1 = llatl * math.pi / 180.
lat2 = llat2 * math.pi / 180.
longl = llongl * math.pi / 180.
long2 = llong2 * math.pi / 180.
cl1 = math.cos(latl)
cl2 = math.cos(lat2)
sl1 = math.sin(latl)
sl2 = math.sin(lat2)
delta = long2 - longl
cdelta = math.cos(delta)
sdelta = math.sin(delta)
y = math.sqrt(math.pow(cl2*sdelta,
2)+math.pow(cll*sl2-sl1*cl2*cdelta, 2))
x = sll * sl2 + cll * cl2 * cdelta
ad = math.atan2(y, x)
dist = ad * rad
obj_coor_y = 0
obj_coor_x = dist/2
target_coor_y = dist
target_coor_x = dist/2
a = 0
j = 0
nestonks = {(0, 0)}
nestonks. remove((0, 0))
nestonks2 = [(0; 0)]
nestonks2.remove((0, 0))
gang = []
dox = target_coor_x - obj_coor_x
doy = target_coor_y - obj_coor_y
speed = 20
done = 1
spid = 1
real_tangle = 8
bi2=0
bi4=0
direction = 0_# 0 - право, 1 - лево
bism = 0 # ноль - маленький на 1 градус, 1 - на 90
ms=211
xs=np.zeros(10)
ys=np.zeros(10)
xin1 = [(0, 0)]
xinl.remove((0, 0))
yinl = [(0, 0)]
yinl.remove((0, 0))
xin2 = [(0, 0)]
xin2.remove((0, 0))
yin2 = [(0, 0)]
yin2.remove((0, 0))
di = [(0, 0)]
di.remove((0, 0))
yi = [(0, 0)]
yi.remove((0, 0))
hits = (0, 0)
hits.discard((0, 0))
navx = dist/2
navy = 0
real_obj_coor_y = 0
real_obj_coor_x = dist/2
tangle = math. pi/2
hangle = math. pi/2
y=0
dt=0
gyroSP=0.01
gyroHY=0.00001
dt1= 0
dt6= 0
def obstacles():
  c = 0
  global nestonks
  for i in range(400):
    xe = round(dist/2 - 200 + c)
    xel = round(dist/2 - 200 + c)
    xe2 = round(dist/2 - 200 + c)
    c = c + l
    v = 0
    for i in range(250):
    ye = 20000+v
    ye1 = 40000+v
    ye2 = 55000+v
    v = v + 1
    di.append(xe)
    yi.append(ye)
    di.append(xe1)
    yi.append(yel)
    di.append(xe2)
    yi.append(ye2)
    nestonks.add((xe2, ye2))
    nestonks.add((xel, yel))
    nestonks.add((xe, ye))
def irove(spid):
  global speed
  if speed < 15:
    speed = speed + spid
  else:
    speed = 20
def stop(spid, bi2):
  global speed
```

```
        global done
        global t
        a = speed
        for i in range(a):
        t = t + 1
        if bi2 == 1 and speed > 0:
        speed = speed - spid
        elif bi2 == 1 and speed == 0:
           print("stopped")
           done = 1
     def Ugolek(bi4, direction, bism):
        global tangle
        global hangle
        a = math.pi / 180
        if bi4 = = 1:
           if bism = = 1:
             a = math.pi / 2
             if direction = = 0:
                hangle = hangle + a
                tangle = tangle + a
             else:
                hangle = hangle - a
                tangle = tangle - a
     def Cam(v1, v2):
        ybasa = 1
        xbasa = 1
        xbasa2 = vl
        ybasa2 = v2
        hangle2 = hangle - math.pi/2
        c = 0
        for i in range(20):
           xes  =  xbasa - 10+c
           c = c + 1
           v = 0
        for i in range(20):
           yes = ybasa + v
           v = v + 1
           real_xes = round(math.cos(hangle2) * xes+
math.sin(-hangle2) * yes+xbasa2)
           real_yes = round(math.sin(hangle2) * xes+
math.cos(hangle2) * yes+ybasa2)
           nestonks2.append((real_xes, real_yes))
        c=0
        for i in range(40):
           xesl = xbasa-20+c
           c=c+1
           v=0
           for i in range(40):
              yes1= ybasa+20+v
              v = v + 1
              real_xesl = round(math.cos(hangle2)*
xesl+irath.sin(-hangle2) * yesl+xbasa2)
              real_yesl = round(math.sin(hangle2)*
xesl+irath.cos(hangle2) * yesl+ybasa2)
              nestonks2.append((real_xesl, real_yesl)) c - 0
for i in range(60):
              xes2 = xbasa - 30 * c
           c =0
           for i in range(60):
           xes2=xbasa-30+c
           c=c+1
           v=0
           for j in range(60):
              yes2  = ybasa+60+v
              v = v + 1
              real_xes2 = round(math.cos(hangle2) * xes2
+math.sin(-hangle2) * yes2+xbasa2)
              real_yes2 = round(math.sin(hangle2) * xes2
+irath.cos(hangle2) * yes2+ybasa2)
              nestonks2.append((real_xes2, real_yes2))
        for i in range(len(nestonks2)):
           if nestonks2[i] in nestonks:
              gang.append(nestonks2[i])
     def ots():
        global gang
        global real_obi_coor_X, prlev, bi5
        if gang != []:
        gang.sort()
        nasr = gang[0]
        xxin = nasr[0]
        if xxin > real_obj_coor_X:
           prlev = 1
        else:
            prlev = 0
            bi5=1
        else:
           bi5  =0
     def exai():
        global t,  real_obj_coor_x, real_obj_coor_y, navX,
navV, tangle
        if bi5 == 1:
           Ugolek(1,  prlev, 1)
           for j in range(15):
              move(5)
              t = t + 1
              coeftr = random.randint(5, 10)
              real_obj_coor_X = real_obj_coor_x
              real_ob1_coor_V = real_ob1_coor_y
              navx = navx + speed * math.cos(hangle)
              navy = navy + speed * math.sin(hangle)
              prlev2 = abs(prlev - 1)
              xinl.append(real_obj_coor_x)
              yinl.append(real_obj_coor_y)
              xin2.append(navx)
              yin2.append(navy)
              Ugolek(1, prlev2, 1)
        for j in range(25):
           move(5)
           t = t * 1
           coeftr = random.randint(5, 10)
           real_obj_coor_x = real_obj_coor_x + speed *
math.cos(tangle)*(1-coeftr/1000)
           real_obj_coor_y = real_obj_coor_y + speed *
math.sin(tangle)*(1-coeftr/1000)
           navx = navx + speed * math.cos(hangle)
           navy = navy + speed * math.sin(hangle)
           prlev2 = abs(prlev - 1)
           xinl.append(real_obj_coor_x)
           yinl.append(real_obj_coor_y)
           xin2,append(navx)
           yin2.append(navy)
        Ugolek(1, prlev2, 1)
        for j in range(15):
           move(5)
           t = t + 1
           coeftr = random.randint(5, 19)
           real_obj_cogr_x = real_obj_coor_x + speed *
math.cos(tangle)* (1 - coeftr /1000)
           real_obj_coor_y = real_obj_coor_y + speed *
math.sin(tangle)* (1 - coeftr /1000)
           navx = navx + speed * math.cos(hangle)
           navy = navy + speed * math.sin(hangle)
           xinl.append(real_obj_coor_x)
           yinl.append(real_obj_coor_y)
           xin2.append(navx)
           Vin2.append(navy)
        Ugolek(1, prlev, 1)
        for i in range(15):
           move(5)
           t = t*1
```

```
            coeftr = random.randint(5, 10)
            real_obj_coor_x = real_obj_coor_x + speed *
math.cos(tangle)* (1 - coeftr /1000)
            real_obj_coor_y = real_obj_coor_y + speed *
math.sin(tangle)* (1 - coeftr /1000)
            navX = navX + speed * math.cos(hangle)
            navy = navy + speed * math.sin(hangle)
            xinl.append(real_obj_coor_x)
            yinl,append(real_obj_coor_y)
            xin2.append(navx)
            yin2.append(navy)
        def shoot():
            global real_obj_coor_x, real_obj_coor_y, navx,
navy, tangle, real_tangle, hangle
            global mtarget coor_x, mtarget coor_y
            a = 0
            j=0
            mtarget_coor_y = (dist, dist, dist, dist-58, dist+58)
            mtarget_coor_x = (dist/2 - 50, dist/2 + 58, dist/2,
dist/2, dist/2)
            for i in range(10):
                target coor_y = mtarget_coor_y[a]
                target coor_x = mtarget_coor_x[a]
                rl = random.randint(-500, 500)
                ri = ri /100
                r2 = random.randint(-30, 30)
                r2 = r2 /1000
                r3 = random.randint(-2, 2)
                r3 = r3 /30
                if (j>1):
                    j=0
                    a=a+1
                dx = target_coor_x - navx
                dy = target_coor_y - navy
                stangle = math.atan2(dy, dx)
                cel = math.hypot(dx, dy)
                sin2a =  cel*9.8/(ms*ms)
                print(sin2a)
                print(cel)
                rnms = ms + rl
                ralf = math.asin(sin2a)/2 + r2
                rbet = stangle +r3
                rtarg_y = rea1_ob1_coor_y+ (math.sin(ralf*2)
* rnms*rnms/9.8) * math.sin(rbet)
                rtarg_x = real_obj_coor_X+ (math.sin(ralf*2) *
rnms*rnms/9.8) * math.cos(rbet)
                rtarg_x = round(rtarg_x)
                rtarg_y = round(rtarg_y)
                xs[i]=(rtarg_x)
                ys[i]=(rtarg_y)
                hits.add((rtarg_x, rtarg_y))
        def exai2():
            global t, real_obj_coor X, real_oDj_coor_V,
navX, navV, tangle
            if bi5 == 1:
                Ugolek(1, prlev, 1)
                for i in range(15):
                    move(5)
                    t = t + 1
                    coeftr = random.randint(5, 10)
                    real_obj_coor_x = reai_obj_coor_x + speed *
irath.cos(tangle) * (1 - coeftr / 1000)
                    real_obj_coor_y = reai_obj_coor_y + speed *
irath.sin(tangle) * (1 - coeftr / 1000)
                    navx = navx+ speed * math.cos(hangle)
                    navy = navy + speed *math.sin(hangle)
                    prlev2 = abs(prlev - 1)
                    xin3.append(real_obj_coor_x)
                    yin3.append(real_obj_coor_y)
                    xin4.append(navx)
                    yin4.append(navy)
                Ugolek(1, prlev2, 1)
                for i in range(25):
                    move(5)
                    t = t + 1
                    coeftr = random.randint(5, IS)
                    real_obj_coor_x = reai_obj_coor_x + speed *
math.cos(tangle) *(1 - coeftr / 1000)
                    real_obj_coor_y = reai_obj_coor_y + speed *
math.sin(tangle) *(1 - coeftr / 1000)
                    navx = navx + speed * irath.cos(hangle)
                    navy = navy + speed * irath.sin(hangle)
                    prlev2 = abs(prlev - 1)
                    xin3.append(real_obj_coor_x)
                    yin3.append(real_obj_coor_y)
                    xin4.append(navX)
                    yin4.append(navy)
                Ugolek(1, prlev2, 1)
                for i in range(15):
                    move(5)
                    t = t + 1
                    coeftr = random.randint(5, 10)
                    real_obj_coor_x = real_obj_coor_x + speed *
math.cos(tangle) * (1 - coeftr / 1000)
                    real_obj_coor_y = real_obj_coor_y + speed *
math.sin(tangle) * (1 - coeftr / 1000)
                    navx = navx + speed * math.cos(hangle)
                    navy = navy + speed * math.sin(hangle)
                    xin3.append(real_obj_coor_x)
                    yin3.append(real_obj_coor_y)
                    xin4.append(navx)
                    yin4.append(navy)
                Ugolek(1, prlev, 1)
                for i in range(15):
                    move(5)
                    t = t + 1
                    coeftr = random.randint(5, 10)
                    real_obj_coor_x = real_obj_coor_x + speed *
math.cos(tangle) * (1 - coeftr / 1000)
                    real_obj_coor_y = real_obj_coor_y + speed *
math.sin(tangle) * (1 - coeftr / 1000)
                    navX = navx +speed * math.cos(hangle)
                    navV = navy +speed * math.sin(hangle)
                    xin3.append(real_obj_coor_x)
                    yin3.append(real_obj_coor_y)
                    xin4.append(navx)
                    yin4.append(navy)
        newx = real_obj_coor_x
        newy = real_obj_coor_y
        newy = tangle
        newx1 = navx
        newy1 = navy
        newT1 = hangle

        obstacles()
        t = 0
        tmax = 5000
        while t < tmax:
            t = t + 1
            coeftr = random.randint(5, 28)
            real_obj_coor_x = real_obj_coor_x + speed *
math.cos(tangle) * (1 - coeftr / 100)
            real_obj_coor_y = real_obj_coor_y + speed *
math.sin(tangle) * (1 - coeftr / 100)
            navX = navx + speed * math.cos(hangle)
            navV = navy + speed * math.sin(hangle)
            Cam(real_obj_coor_x, real_obj_coor_y)
            ots()
            exai()
            nestonks2 = [(0, 0)]
```

```
            nestonks2.remove((0, 0))
            gang = []
            if dt == 30:
               dt = 0
               Kamushki = random.randint(-5, 5)
               yamochki = random.randint(-5, 5)
               zinak = random.randint(0, 1)
               rgrs = random.randint(967, 1053)
               rgry = random.randint(967, 1053)
               if zinak == 0:
                 zinak = - 1
               real_obj_coor_x = real_obj_coor_x + speed *
math.cos(tangle) * (1- kamushki / 1000) * (1 - yamochki
/ 1000)
               real_obj_coor_y = real_obj_coor_y + speed *
math.sin(tangle) * (1- kamushki / 1000) * (1 - yamochki /
1000)
               tangle = tangle + tangle * zinak * (1 -
kamushki / 10000) * (l - yamochki/10000) / 100
               drx = real_obj_coor_x - newx
               dry = real_obj_coor_y - newy
               drT = newT - tangle
               drx1 = navx - newx1
               dry1 = navy - пewy1
               drT1 = newT1 - tangle
               gyroSP  = (drx - drxl) * rgrs / 1000
               gyroSPy = (dry - dry1) * rgrs / 1000
               gyroHY  = (drt - drtl) * rgry / 1000
               hangle  = hangle + gyroHY
               navx = navx + math.cos(hangle) * gyroSP
               navy = navy + math.sin(hangle) * gyroSPY
               newx = real_obj_coor_x newy =
real_obj_coor_y
               newT = tangle
               newx1l = navx newyl = navy
               newTl = hangle
            else:
               dt = dt + 1
               xinl.append(real_obj_coor_x)
               yinl.append(real_obj_coor_y)
               xin2.append(navx)
               yin2.append(navy)
               dox = target_coor_x - navx
               doy = target_coor_y - navy
               EL = math.hypot(dox, doy)
               tangle = math.acas(dox / EL)
               if math.hypot(dox, doy) > 1000
                 if dt6 == 40:
                 dt6 = 0
                 hangle = ttangle
                 tangle = hangle
               else:
                 dt6 = dt6 + 1
               else:
                  break
         xin3 = [(0, 0)]
         xin3.remove((0, 0))
         yin3 = [(0, 0)]
         yin3.remove((0, 0))
         xin4 = [(0, 0)]
         xin4.remove((0, 0))
         yin4 = [(0, 0)]
         yin4.remove((0, 0))
         t = 0
         target_coor_x = dist/2
         target_coor_y = 0
         tangle = tangle + math.pi
         hangle = hangle + math.pi
         while t < tmax:
            t = t + 1
               coeftr = random.randint(5, 20)
               real_obj_coor_x = real_obj_coor_x + speed *
math.cos(tangle) * (1 - coeftr / 100)
               real_obj_coor_y = real_obj_coor_y + speed *
math.sin(tangle) * (1 - coeftr / 100)
               navx = navx + speed * math.cos(hangle)
               navy = navy + speed * math.sin(hangle)
               Cam(real_obj_coor_x, real_obj_coor_y)
               ots()
               exai2()
               nestonks2 = [(0, 0)]
               nestonks2.remove((0, 0))
               gang = []
               if dt == 30:
                 dt = 0
                 Kamushki = random.randint(-5, 5)
                 yamochki = random.randint(-5, 5)
                 zinak = random.randint(0, 1)
                 rgrS = random.randint(985, 1055)
                 rgr^ = random.randint(985, 1055)
                 if zinak == 0:
                 zinak = - 1
                 real_obj_coor_x = real_obj_coor_x + speed *
math.cos(tangle) * (1 - kamushki / 1000) * (1 - yamochki
/ 1000)
                 real_obj_coor_y = real_obj_coor_y + speed *
math.sin(tangle) * (1 - kamushki / 1000) * (1 - yamochki
/ 1000)
                 tangle = tangle + tangle * zinak * (1 -
kamushki / 10000) * (1 - yamochki / 10000) / 100
                 drx = real_obj_coor_x - newx
                 dry = real_obj_coor_y - newy
                 drT = newT - tangle
                 drx1 = navx - newx1
                 dryl = navy - newyl
                 drTl = newT1 - tangle
                 gyroSP = (drx - drx1) * rgrs / 1000
                 gyroSPY = (dry - dry1) * rgrs / 1000
                 gyroHV = (drT - drTl) * rgry / 1000
                 hangle = hangle + gyroHY
                 navX = navX + math.cos(hangle) * gyroSP
                 navV = navV - math.sin(hangle) * gyroSPV
                 newX = real_obj_coor_x
                 newV = real_obj_coor_y
                 newT = tangle
                 newxl = navx
                 newyl = navy
                 newTl = hangle
               else:
                 dt = dt + 1
               xin3.append(real_obj_coor_x)
               yin3.append(real_obj_coor_y)
               xin4.append(navx)
               yin4.append(navy)
               dox = target_coor_x - navx
               doy = target_coor_y - navy
               EL = math.hypot(dox, doy)
               rgf = random.randint(990, 1020)
               ttangle = math.atan2(doy, dox) * rgf/1000
               if math.hypot(dox, doy) > 100:
                 if dt6 == 40:
                     dt6 = 0
                     hangle = ttangle
                     tangle = hangle
                  else:
                  dt6 = dt6 + l
               else:
                   break
            print(real_obj_coor_x, real_obj_coor_y)
            x3 = di
```

```
    уз = yi
    x2 = xin2
    y2 = yin2
    xl = xinl
    yl = yinl
    x4 = xin3
    y4 = yin3
    fig, ax = plt.subplots(figsize=(9, 7))
    ax.set.title('путь, fontsize=16)
    ax.set_xlabel(nx координата"; fontsize=14)
    ax.set_ylabel(ny координата"; fontsize=14)
    ax.grid (which="major", linewidth=1.2)
    ax.grid (which="minor", linestvle="--",
color="gray", linewidth=0.5)
    ax.scatter(xl, yl, o="red")
    ax.scatter(x2, y2, c="blue")
    ax.scatter(x3, y3, c="green")
    ax. scatter(x4, y4, z="yellow")
    ax.scatter(x5, y5, :="grey")
    ax.xaxis.set_minor_locator(AutoMinorLocator())
    ax.yaxis.set_minor_locator(AutoMinorLocator())
    ax.tick_params(which='major', length=18, width=2)
    ax.ticK_params(which='minor', length=5, width=1)
    plt.xlim(dist*S.3/2, dist*1.2/2)
    plt.ylim(0, dist*l.l)
    plt.show()
    x = (xs[0], xs[l], xs[2], xs[3], xs[4], xs[5], xs[6],
xs[7], xs[8], xs['
    V = (ys[S], ys[l], ys[2], ys[3], ys[4], ys[5], ys[6],
ys[7], ys[8], ys['
    xll = (dist /2-50, dist /2+50, dist / 2, dist / 2, dist / 2)
    yll = (dist, dist, dist, dist - 50, dist + 50)
    fig, ax = plt.subplots(figsize=(9, 7))
    ax.set_title("Места попаданий выстрелов и зоны
поражения", fontsize=16)
    ax.set_xlabel("X координата", fontsize=14)
    ax.set_ylabel("Y координата", fontsize=14)
    ax. grid (which=" major", linewiflth=1.2)
    ax. grid (which=" minor", linestvles" ┐",
color="gray", linewiflth=0.5)
    ax.scatter(x, y, c="red")
    circlel = plt.Circle((x[0], y[0]), 30, colon='red',
fill=False)
    circle2 = plt.Circle((x[l], y[l]), 30, colors'red1,
fill=False)
    circle3 = plt.Circle((x[2], y[2]), 30, colors'red1,
fill=False)
    circle4 = plt.Circle((x[3], y[3]), 30, colors'red1,
fill=False)
    circle5 = plt.Circle((x[4], y[4]), 30, colors'red1,
fill=False)
    circle6 = plt.Circle((x[5], y[5]), 30, colors'red1,
fill=False)
    circle7 = plt.Circle((x[6], y[6]), 30, colors'red1,
fill=False)
    circle8 = plt.Circle((x[7], y[7]), 30, colors'red1,
fill=False)
    circle9 = plt.Circle((x[3], y[a]), 30, color='red',
fill=False)
    circle0 = plt.Circle((x[9], y[9]), 30, colon='red',
fill=False)
    ax. scatter (xll, yll, : = "blue", marker=r'x")
    circles = plt.Circle((xll[2], yll[2]), 250, colon-'blue1,
fill=False)
    ax.xaxis.set_minor_locator(AutoMinorLocator())
    ax.yaxis.set_minor_locator(AutoMinorLocator())
    ax.tick_params(which='major1, length=18, wiflth=2)
    ax.tick_params(i'jhich='minor1, length=5, width=l)
    ax.add_artist(circlel)
    ax.add_artist(circle2)
    ax.add_artist(circle3)
    ax.add_artist(circle4)
    ax.add_artist(circle5)
    ax.add_artist(circle6)
    ax.add_artist(circle7)
    ax.add_artist(circle8)
    ax.add_artist(circle9)
    ax.add_artist(circle10)
    ax.add_artist(circle0)
    plt.xlim(dist*0.9/2, dist*l.l/2)
    plt.ylim(dist*0.95, dist*l.l)
    plt.show()
```

Алексей Иванович Данилко – студент Московского института электроники и математики Национального исследовательского университета «Высшая школа экономики», г. Москва,
E-mail: aidanilko@miem.hse.ru

Олег Владимирович Стукач – доктор технических наук, профессор Московского института электроники и математики Национального исследовательского университета «Высшая школа экономики» и Новосибирского государственного технического университета, основатель Томской группы Института инженеров по электротехнике и радиоэлектронике IEEE.
E-mail: tomsk@ieee.org

Статья поступила 25.02.2022

Simulation of the Odometric Autonomous Navigation System of a Crawler Robot Optimal in Performance Criteria

A.I. Danilko[1], O.V. Stukach[1,2]

[1]National Research University "Higher School of Economics", Moscow, Russia
[2]Novosibirsk State Technical University, Novosibirsk, Russia

Abstract: The navigation system of the autonomous moving crawler vehicle in the open area is simulated. The possibility of using odometry as a navigation system for the robot when performing a task and return to the starting position is considered. The designed software is given. An overview of other solutions for the navigation of robots is given, the possibility of using them as an additional tool to improve the accuracy of determining coordinates is defined. An algorithm for converting geographical coordinates into a new coordinate plane has been obtained. An

algorithm for finding a route to the target, a mechanism for calculating the current own position based on completed movements with a mechanism for making corrections to errors based on data received from some peripheral device, such as a gyroscope, and an algorithm for returning to the point of departure has been designed. The description of the software modules implementing the selected solutions is given, its effectiveness is evaluated.

Keywords: robotics, software control, odometry, coordinate define, the robot navigation.

REFERENCES

[1] IFR presents World Robotics Report 2020 – International Federation of Robotics [Electronic resource]. URL: https://ifr.org/ifr-press-releases/news/record-2.7-million-robots-work-in-factories-around-the-globe?utm_source=CleverReach&utm_medium=email&utm_campaign=NL+3%2F20&utm_content=Mailing_12155770 (accessed: 07.03.2021).

[2] Analiz opyta boevogo primeneniya grupp bespilotnykh letatelnykh apparatov dlya porazheniya zenitno-raketnykh kompleksov sistemy protivovozdushnoi oborony v voennykh konfliktakh v Sirii, v Livii i v Nagornom Karabakhe (Analysis of the experience of application of unmanned aerial vehicles groups to defeat anti-aircraft missile systems of the air defense system in military conflicts in Syria, Libya and Nagorno-Karabakh [Electronic resource]. URL: https://www.elibrary.ru/item.asp?id=44437305 (accessed: 07.03.2021).

[3] Moskvin I., Lavrenov R., Magid E., Svinin M., ”Modelling a Crawler Robot Using Wheels as Pseudo-Tracks: Model Complexity vs Performance” IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA). 2020, Pp. 1–5, Doi: 10.1109/ICIEA49774.2020.9102110. – https://ieeexplore.ieee.org/document/9102110

[4] Ingle S., Phute M., “Tesla Autopilot: Semi Autonomous Driving, an Uptick for Future Autonomy” Int. Res. J. Eng. Technol. 2016, vol. 3, no. 9.

[5] Yan W., Weber C., Wermter S., "Learning indoor robot navigation using visual and sensorimotor map information", Front. Neurorobot, 2013, vol. 7, no. OCT.

[6] Ruiz-Sarmiento J.R., Galindo C., Gonzalez-Jimenez J., "Building Multiversal Semantic Maps for Mobile Robot Operation", Knowledge-Based Syst, 2017, vol. 119.

[7] "Pattern Recognition and Machine Learning", J. Electron. Imaging, 2007, vol. 16, no. 4.

[8] Li X. et al. "A visual navigation method of mobile robot using a sketched semantic map", Int. J. Adv. Robot. Syst, 2012, vol. 9.

[9] Yu X., Marinov M. "A study on recent developments and issues with obstacle detection systems for automated vehicles", Sustainability (Switzerland), 2020, vol. 12, no. 8.

[10] "Razrabotka sistemy avtonomnoi navigatsii na osnove tekhnologii lidarnogo skanirovaniya mestnosti i software defined radio (The design of autonomous navigation system based on technologies of lidar scan and software defined radio)". International Research Journal, 2017, no. 7–3 (61).

[11] Wang D., Watkins C., Xie H. "MEMS mirrors for LiDAR: A review", Micromachines, 2020, vol. 11, no. 5.

[12] Debeunne C., Vivet D. "A review of visual-lidar fusion based simultaneous localization and mapping", Sensors (Switzerland), 2020, vol. 20, no. 7.

[13] Bai Y., Wang Y., Svinin M., Magid E., Sun R. "Adaptive Multi-Agent Coverage Control With Obstacle Avoidance", IEEE Control Systems Letters, 2022, vol. 6, pp. 944–949, doi:10.1109/LCSYS.2021.3087609. – https://ieeexplore.ieee.org/document/9448334.

[14] Zhan W. et al, "Autonomous visual perception for unmanned surface vehicle navigation in an unknown environment", Sensors (Switzerland), 2019, vol. 19, no. 10.

[15] Gu J. et al., "Recent advances in convolutional neural networks", Pattern Recognit, 2018, vol. 77.

[16] Kononov K., Lavrenov R., Tsoy T., Martiez-Garc E.A., Magid E., "Virtual Experiments on Mobile Robot Localization with External Smart RGB-D Camera Using ROS", IEEE 3rd Eurasia Conference on IOT, Communication and Engineering (ECICE), 2021, pp. 656–659, doi: 10.1109/ECICE52819.2021.9645644. – https://ieeexplore.ieee.org/document/9645644

[17] Roth W. et al., "Resource-efficient neural networks for embedded systems" arXiv. 2020.

[18] Shukla D., Jha R.K., "Robust motion estimation for night-shooting videos using dual-accumulated constraint warping" J. Vis. Commun. Image Represent, 2016, vol. 38.

[19] Genzel D., Yovel Y., Yartsev M.M., "Neuroethology of bat navigation", Current Biology, 2018, vol. 28, no. 17.

[20] Crockett A., Rueckner W., "Visualizing sound waves with schlieren optics", Am. J. Phys, 2018, vol. 86, no. 11.

[21] Sligar A.P., "Machine learning-based radar perception for autonomous vehicles using full physics simulation", IEEE Access, 2020, vol. 8.

[22] Milner G., "What is GPS?" J. Technol. Hum. Serv., 2016, vol. 34, no 1.

[23] Yasyukevich Yu.V. et al., "Vliyanie differentsialnykh kodovykh zaderzhek GPS/GLONASS na tochnost opredeleniya absolyutno polnogo elektronnogo soderzhaniya ionosfery" (The influence of differential code delays of GPS/GLONASS on the accuracy of determining the absolute total electronic content of the ionosphere). In Geomagnetism and Aeronomy, 2015, vol. 55, no. 6.

[24] Cai H. et al. "Optimization of Maintenance Force Dispatch of Artillery Command Information System Based on Mission Success", Smart Innovation, Systems and Technologies, 2021, vol. 190.

[25] Sorokin I., Obukhov A., Romanov et al., "Application of a permutation decoding method in a small-class unmanned aerial vehicle control system (drones, multicopters)", Mod. Technol. Syst. Anal. Model, 2019, vol. 2(62), no. 2(62), p. 186–195.

[26] Kutuzov D., Osovsky A., Stukach O., Starov D., "Modeling of IIoT Traffic Processing by Intra-Chip NoC Routers of 5G/6G Networks", International Siberian Conference on Control and Communications (SIBCON), 13–15 May 2021, Kazan, Russia, DOI:

10.1109/SIBCON50419.2021.9438874. – https://ieeexplore.ieee.org/document/9438874
[27] Zou Y. et al., "A Survey on Wireless Security: Technical Challenges, Recent Advances, and Future Trends", Proceedings of the IEEE, 2016, vol. 104, no 9.
[28] Manzanilla A. et al., "Autonomous navigation for unmanned underwater vehicles: Real-time experiments using computer vision", IEEE Robot. Autom. Lett., 2019, vol. 4, no. 2.
[29] Zhan H. et al., "Visual Odometry Revisited: What Should Be Learnt?", IEEE International Conference on Robotics and Automation, 2020.
[30] Steinberg J.M., Pelland N.A., Eriksen C.C., "Observed evolution of a California Undercurrent eddy", J. Phys. Oceanogr, 2019, vol. 49, no. 3.
[31] Tian Q., Wang K.I.K., Salcic Z., "An INS and UWB fusion-based gyroscope drift correction approach for indoor Pedestrian tracking", Sensors (Switzerland), 2020, vol. 20, no. 16.
[32] Marinushkin P.S., Bakhtina V.A., Podshivalov I.A., Stukach O.V., "Voprosi razrabotki inertsialnykh peshekhodnykh navigatsionnykh sistem na osnove MEMS-datchikov" (Issues of design of the inertial pedestrian navigation systems based on MEMS sensors), Science and Education. Electronic edition. FGBOU VPO "Bauman Moscow State Technical University", 2015, no. 06, p. 157–173. DOI: 10.7463/0615.0786740, http://technomag.bmstu.ru/doc/786740.html.
[33] Yuan M., Yau W.Y., Li Z., "Lost Robot Self-Recovery via Exploration Using Hybrid Topological-Metric Maps", IEEE Region 10 Annual International Conference, Proceedings/TENCON, 2019, vol. 2018, October.
[34] Rusyak I.G., Tenenev V., "Modeling of ballistics of an artillery shot taking into account the spatial distribution of parameters and backpressure", Comput. Res. Model, 2020, vol. 12, no. 5.
[35] Bogatinov S.V., "Effektivnost vypolneniya ognevoi zadach – kriterii otsenki (The effectiveness of the fire task - evaluation criterion)", Proceedings of the Russian Academy of Rocket and Artillery Sciences, 2008, no. 1 (57), p. 38–40.
[36] Bentsen M. et al., "Coordinate transformation on a sphere using conformal mapping", Mon. Weather Rev, 1999, vol. 127, no. 12.
[37] Llavori I. et al., "Critical Analysis of Coefficient of Friction Derivation Methods for Fretting under Gross Slip Regime", Tribol. Int., 2020, vol. 143.
[38] Park S., "Adaptive control of a vibratory angle measuring gyroscope", Sensors, 2010, vol. 10, no. 9.
[39] Fu H., Yu R., "LIDAR scan matching in off-road environments", Robotics, 2020, vol. 9, no. 2.
[40] Yu B. et al., "Optimization and testing of suspension system of electric mini off-road vehicles", Sci. Prog, 2020, vol. 103, no. 1.
[41] 41. Musalimov V.M., Rotts Yu.A., Astafyev S.A., Amvrosyeva A.V., "Raschet nadezhnosti uprugikh elementov mikromekhanicheskikh giroskopov" (Calculation of reliability of elastic elements of micromechanical gyroscopes), [Electronic resource], URL: https://www.studmed.ru/musalimov-v-m-rotc-yu-a-astafev-s-a-amvroseva-a-v-raschet-nadezhnosti-uprugih-elementov-mikromehanicheskih-giroskopov_a819c2f85c4.html (accessed: 06.05.2021).

Alex I. Danilko is student at Moscow Institute Electronics and Mathematics of National Research University Higher School of Economics, Moscow, E-mail: aidanilko@miem.hse.ru

Oleg V. Stukach is the founder of the Tomsk IEEE Chapter, Dr. of Sci., Professor of Moscow Institute Electronics and Mathematics of National Research University Higher School of Economics and Novosibirsk State Technical University.
E-mail: tomsk@ieee.org

The paper has been received on 25/02/2022