Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем

Системное программирование

Гальковский Антон Денисович

Автономная 3D навигация

Выпускная квалификационная работа бакалавра

Научный руководитель:
д.ф.-м.н., проф. А.Н. Терехов

Консультант:
ст.преп. А.А. Пименов

Рецензент:
к.ф.-м.н. К.С. Амелин

Санкт-Петербург
2020

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems

System Programming

Anton Galkovsky

Autonomous 3D navigation

Bachelor's Thesis

Scientific supervisor:
Professor Andrey Terekhov

Consultant:
Senior lecturer Alexander Pimenov

Reviewer:
Ph.D. of Physico-mathematical Sciences Konstantin Amelin

Saint-Petersburg
2020

Оглавление

Введение

В течение последнего десятилетия произошло существенное продвижение в сфере робототехники летательных аппаратов и, в частности, квадрокоптеров. Основные примеры их использования включают съёмку видеозаписей с недоступных человеку ракурсов, построение карты местности, транспортировку вещей и исследование труднодоступных или опасных для человека объектов. Почти в каждом из сценариев использования квадрокоптера так или иначе приходится решать проблему избежания столкновений с препятствиями. В случае, если робот является автономным (не подразумевает прямого управления человеком), эта проблема становится особенно актуальной. Более того, при автономном движении робота должна учитываться возможность потери сигнала GPS и других сетей. По этой причине в "базовый" набор сенсоров входят только IMU (Inertial Measurement Unit) и некоторая система видеозрения, в роли которой могут выступать группа камер или лидар. Однако стоимость лидара обычно в десятки раз превышает стоимость камеры, поэтому использование группы камер всё же оказывается дешевле, хоть и сложнее с технической точки зрения.

На данный момент существует всего несколько промышленных решений, ограничивающихся группой камер и IMU и действительно производящих реконструкцию окружающего мира во время навигации. Хорошим примером может быть Skydio 2 [6], использующий 6 камер, в частности, для обзора всего окружающего пространства. Однако, с другой стороны, человек может без особых проблем управлять квадрокоптером, используя изображение всего лишь с одной камеры. В этом контексте возникает вопрос: будет ли достаточно одной камеры для автономной навигации квадрокоптера? Этот случай оказывается концептуально более сложным из-за следующей проблемы.

Реконструкция карты окружающего мира по нескольким снятым изображениям обычно производится посредством отслеживания смещений точек на попавших в кадр объектах. Если же какой-то объект окажется близко к касательной к траектории квадрокоптера, то в пре-

делах этой части траектории его точки будут смещаться очень мало на разных снимках, из-за чего расстояние до этого объекта будет восстанавливаться с большой погрешностью. Это может привести к ухудшению качества навигации, поскольку построение маршрута будет основываться на искажённых данных о карте. Более того, если при поступательном движении робота какой-то небольшой объект окажется точно на прямой его траектории, он может быть вообще не распознан, что приведёт к столкновению.

С этой позиции возникает потребность в отслеживании того, каким областям пространства могут принадлежать те или иные точки. Во-первых, это бы позволило точнее локализовывать точки, поскольку информация об их положении сохранялась бы за всё время, которое они были видны, в отличие от стандартного способа, при котором координаты точки вычисляются на основании некоторого количества последних снятых изображений. Во-вторых, появилась бы возможность явно отслеживать ситуации, при которых погрешность восстановления положения точек оказывается слишком большой для обеспечения безопасности движения.

С другой стороны, информация об областях возможного расположения точек карты помогла бы изменять при необходимости составленную траекторию, добавляя в неё сдвиги в поперечном направлении, чтобы на снимках камеры начали смещаться даже ранее неподвижные точки, а значит, повысилась бы точность их локализации.

1. Постановка задачи

Целью данной работы является разработка подхода к навигации квадрокоптеров, основанного на отслеживании областей возможного положения точек в пространстве и использующего идею с поперечными сдвигами для локального увеличения числа ракурсов обзора на обнаруженные препятствия для уточнения расположения их точек.

Для этого требуется решить следующие задачи:

- изучение предметной области и поиск подхода к навигации для взятия за основу,
- внесение в его реализацию необходимых изменений для начала реализации предлагаемого подхода,
- поиск и реализация метода отслеживания областей возможного положения точек,
- реализация идеи с поперечными сдвигами,
- анализ влияния предлагаемого подхода на работоспособность алгоритма навигации.

2. Обзор

2.1. Системы видеозрения

Как упоминалось во введении, система видеозрения, используемая для автономной навигации, может быть представлена одной либо несколькими камерами или лидаром. Также иногда применяется особый тип камер – rgbd-камеры, которые помимо (или вместо) обычного снимка возвращают карту глубин.

Рассмотрим основные преимущества и недостатки перечисленных сенсоров.

Лидар является одним из самых удобных вариантов для получения карты окружения, поскольку имеет достаточно большую дальность детектирования объектов (порядка 100 метров), а получаемые данные почти не требуют дополнительной обработки для добавления в восстанавливаемую роботом карту. Однако его существенным минусом является высокая в среднем стоимость: для задач автономной навигации обычно используются модели вроде Velodyne Puck [3], и их цена оказывается порядка 5000$ [1].

Rgbd-камера, позволяющая также не очень сложным образом восстанавливать карту окружения, имеет уже сравнительно небольшую стоимость порядка нескольких сотен долларов (пример - [5]), но обладает другим важным недостатком, заключающимся в фиксированной не очень большой дальности обзора (порядка 10 метров). Это делает её использование осмысленным для полётов, например, внутри зданий, но плохо подходит для организации полёта в лесу с большой скоростью.

Обычные камеры не обладают указанными недостатками лидара и rgbd-камеры, но также не позволяют напрямую получать карту окружения. Для этого и приходится решать задачу реконструкции карты по получаемым с камеры снимкам.

2.2. Задача SLAM-а

Помимо планирования траектории робота важной частью процесса автономной навигации является восстановление карты окружения и пройденного в ней пути. Эта задача называется SLAM-ом (Simultaneous Localization And Mapping). При её решении кроме данных с камеры обычно также используется информация, получаемая с IMU. Однако, из-за неизбежного накопления этим датчиком ошибки с течением времени восстанавливать пройденный путь, основываясь только на его данных, нельзя. Поэтому задача позиционирования робота требует для своего решения одновременного анализа данных как с IMU, так и с камеры.

2.3. Критерии выбора подхода

Подход к навигации, реализацию которого требовалось выбрать для взятия за основу, должен был обладать следующими качествами.

- Во-первых, требовалось, чтобы его метод построения траектории был универсальным. Иными словами, в нём не должно было присутствовать слишком специфичных отличий от других методов. Данное требование возникало для того, чтобы предлагаемый в данной работе подход был легко переносим и на другие способы навигации. К примеру, подход [7], предложенный в 2018 году исследователями из Carnegie Mellon University, позволяет достичь скорости полёта в 10 м/с, однако их решение основано на предварительном расчёте альтернативных траекторий движения, что неочевидным образом согласуется с другими методами навигации.

- Во-вторых, выбранный подход должен был сам по себе показывать хорошие характеристики построения траектории по сравнению с другими. Невыполнение этого условия ставило бы под сомнение актуальность данной работы, поскольку даже доказанная с помощью тестов полезность предлагаемого подхода для посредствен-

ного метода навигации не давала бы гарантии того, что он будет полезен для хотя бы одного образцового способа.

- В-третьих, что довольно естественно, он должен был иметь открытую реализацию, которую было бы возможно запустить в виртуальной среде, позволяющей удобным образом производить отладку и наблюдение за процессом навигации.

- В-четвёртых, было желательно, чтобы структура его реализации позволяла удобным образом вносить изменения как в код различных компонентов, так и в неё саму.

2.4. Выбранный подход

Найденный подход, соответствующий перечисленным критериям, описывается группой исследователей университета HKUST в их статье [2], опубликованной в 2018 году.

Его алгоритм построения маршрута, в целом, состоит из двух блоков: поиска безопасного коридора и генерации конкретной траектории.

- Первый блок заключается в построении "каркаса" для будущей траектории. Для этого внутри сохранённой карты генерируются точки с помощью вероятностного алгоритма, причём для каждой из них проверяется удалённость от препятствий на некоторое заранее заданное расстояние. Из точек, соответствующих этому условию, выстраивается дерево безопасных путей, в котором затем выбирается наиболее подходящая ветвь. В результате чего в пространстве получается безопасный коридор, составленный из последовательности соприкасающихся сфер, не содержащих в себе точек препятствий.

- Второй же блок призван сгенерировать такую траекторию робота внутри полученного коридора, которая была бы оптимальна, безопасна и выполнима с точки зрения физики. Для этих целей указанные требования записываются в специальном виде и для них

решается задача конического программирования второго порядка, в результате решения которой получается набор коэффициентов для кривых Безье, из которых и состоит траектория.

Указанные блоки выполняются итеративно: сначала прокладывается коридор до конечной цели полёта, затем второй блок генерирует уже конкретную траекторию, а пока робот пролетает её небольшой участок, дерево из первого блока перестраивается так, чтобы соответствовать новым полученным данным, и так далее.

2.5. Структура системы

Открытая реализация данного подхода была выполнена на C++ для фреймворка ROS. В нём программа представляется в виде набора узлов, общающихся между собой посредством очередей сообщений. Это позволило достаточно прозрачным образом вносить изменения, добавляя новые узлы или заменяя существующие.

Виртуальная среда организовывалась с помощью отрисовки квадрокоптера, карты и других элементов через узел rviz. Для этого некоторым узлам, помимо их основной работы, также необходимо было своевременно посылать обновлённые данные о процессе полёта на отрисовку. Таким образом, контроль за корректностью движений и симуляция работы датчиков также выполнялись на стороне реализации этого подхода. Однако, с другой стороны, это облегчило отладку, так как появлялась возможность следить за всеми протекающими во время навигации процессами.

2.6. Произведённые испытания

Важно отметить, что в виртуальной среде использовался ”идеальный” IMU. Это проявлялось в том, что физические характеристики движения (такие как скорость или ускорение) никак специально не искажались при отправке различным компонентам. Аналогичные особенности (хотя зачастую и более тонкие) не позволяют в обычном случае

гарантировать, что алгоритм навигации, корректно работающий в виртуальном мире, будет также работать в реальном. Из-за этого требуется проводить отдельные тесты уже с использованием реального робота.

Для проверки работоспособности данного подхода был произведён ряд тестов с использованием реального квадрокоптера в различных типах местностей (и в том числе в лесу). В качестве системы видеозрения использовался лидар модели Velodyne Puck LITE [4]. Вычисления производились на процессоре Intel i7–5500U. Результаты тестов также приведены в указанной статье.

2.7. Потребовавшиеся изменения

Первым недочётом реализации выбранного подхода (с точки зрения данной работы) стало то, что в нём в качестве датчика, считывающего информацию об окружающей карте, симулировался лидар. Более того, в его реализации считывались все точки в некотором радиусе, даже те, которые не могут быть видны роботу в данный момент. Однако в данной работе требовалось симулировать именно камеру, чего, ввиду последнего замечания, нельзя было сделать, просто ограничивая угол, в котором считываются точки.

Второй проблемой стало то, что сгенерированная карта хранилась в виде облака точек, что не позволяет напрямую просимулировать работу камеры, поскольку для этого требуется информация о том, как разные объекты загораживают друг друга. Также объекты на карте генерировались в виде групп находящихся рядом столбиков разной высоты, имеющих ширину в один воксель. Такое представление препятствий не приносило никаких дополнительных затрат для существующей симуляции лидара, поскольку в ней считывание ближайших точек в некотором радиусе происходило с помощью запроса на поиск к k-d дереву, хранящему всю карту. Однако при симуляции камеры это привело бы к излишним вычислительным расходам, если, например, представлять каждый из столбиков в виде отдельного параллелепипеда.

3. Внесение необходимых изменений

Таким образом, для исправления проблем, описанных в 2.7, требовалось изменить способ представления и хранения карты, а также заменить симуляцию лидара на симуляцию pinhole-камеры.

3.1. Новое представление карты

Каждое препятствие на карте было решено представлять в виде параллелепипеда, образованного прямоугольными полигонами. Этот способ, во-первых, подходил для симуляции камеры, во-вторых, он хорошо сочетался с существующим способом генерации карты, так как теперь каждая группа столбиков заменялась на один большой параллелепипед, и, в-третьих, позволял избежать лишних затрат на обработку каждого столбика по-отдельности. Вершины полигонов, соответственно, также генерировались в воксельной сетке заданного разрешения.

В результате, настраивая границы возможных значений для высоты и ширины генерируемых параллелепипедов, можно было получить карту, в достаточной степени соответствующую таким типам местности, как сосновый лес, когда параллелепипеды получаются узкими и высокими, представляя стволы деревьев, или же как район города, когда ширина параллелепипедов оказывается сопоставимой с их высотой. Также, в перспективе, такое представление позволяет отображать и более сложные объекты на карте, вручную задавая составляющие их параллелепипеды (при необходимости).

Стоит отметить, что узел rviz, обеспечивающий визуализацию виртуальной среды, в большей степени предназначен для работы с облаками точек. Из-за этого, для отрисовки карты, полученные полигоны требовалось замостить точками, также в соответствии с воксельной сеткой.

Далее появился вопрос о том, каким образом в симуляции камеры будет организовываться получение ключевых точек. Эта проблема была связана с тем, что в реальном мире при реконструкции карты по двум снимкам на них ищутся пары локально похожих пикселей,

которые соответствуют одной и той же хорошо различимой точке на видимом объекте. В текущем же представлении карты все объекты получались ”идеально однотонными”, то есть не имели никаких отметок на своей поверхности.

Для решения этой проблемы было решено после генерации карты добавлять на поверхность полигонов некоторое количество меток, положение которых вычислялось случайным образом. Всё множество меток представлялось в виде облака точек, соответствующего воксельной сетке карты, причём каждая группа меток ассоциировалась со своим полигоном, для того чтобы во время рендеринга можно было легко узнать, какому полигону принадлежит данная группа. Число меток для каждого полигона определялось заранее заданным процентным отношением от общего числа вокселей, замостивших данный полигон.

В конечном счёте, карта начала выглядеть следующим образом (рис. 1):

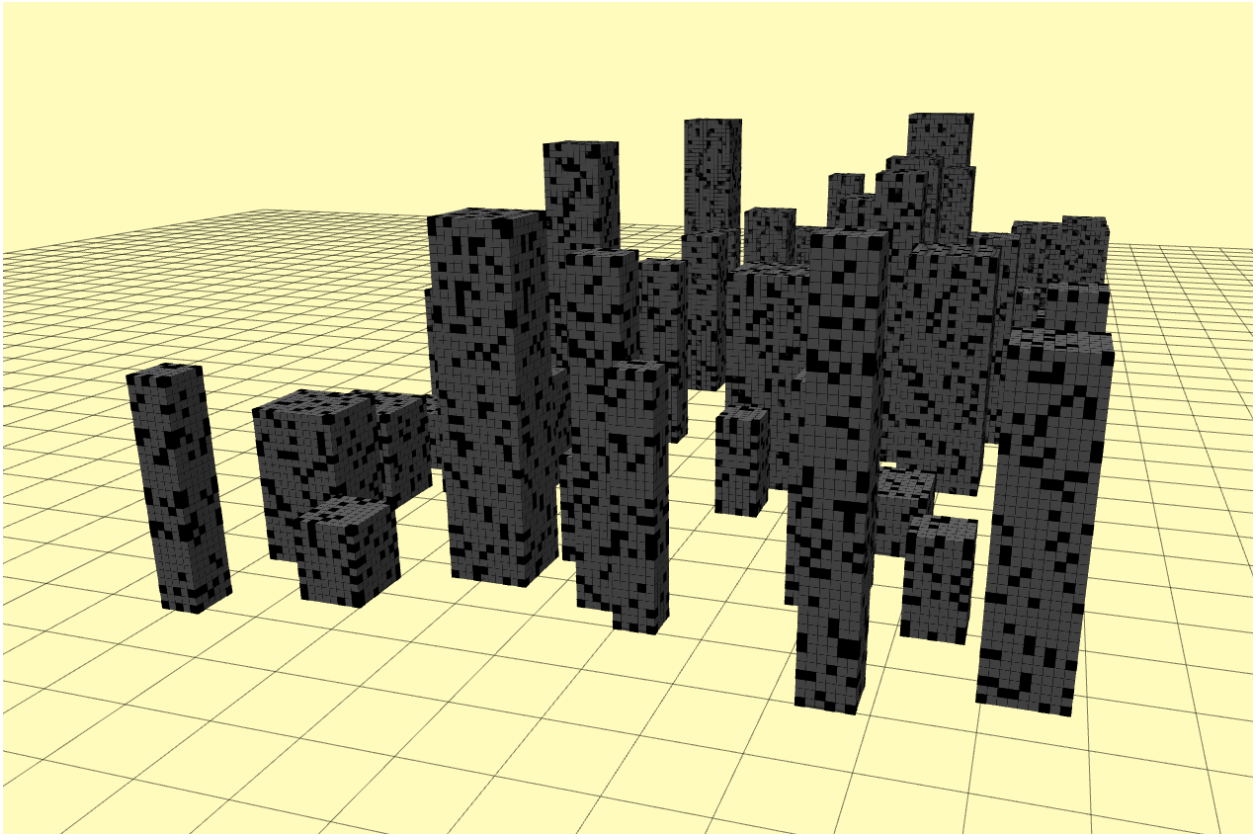

Рис. 1: Новое представление карты, визуализированное узлом rviz

3.2. Симуляция обычной камеры

Для рендеринга карты было решено сделать свою реализацию. Это позволяло, во-первых, поддерживать выбранное представление карты в виде полигонов и ассоциированных с ними групп меток, а во-вторых, упростить реализацию следующих за рендерером компонентов, например, обеспечивая глобальную идентификацию отсканированных меток.

Принцип работы реализованного рендерера имел общую идею с алгоритмом Scanline rendering.

- Сначала происходил перебор всех полигонов на карте. В случае, если полигон был ориентирован правильным образом к камере, а также хотя бы частично входил в её область обзора, происходил расчёт, в какие пиксели изображения спроецировались бы его рёбра. В результате чего для каждого горизонтального ряда пикселей будущего изображения запоминалось, какой его подотрезок соответствует данному полигону.

- Затем каждый ряд проходился один раз слева-направо. В процессе этого прохода отслеживалось, какой из сохранённых (и перекрывающихся) отрезков виден на самом деле. Соответственно, для пройденных пикселей вызывалась функция записи необходимой информации.

Обработка меток основывалась на сохранении информации о том, куда бы спроецировалась каждая из них, при условии того, что соответствующий полигон потенциально мог быть увиден (соответствовал условию из первого пункта).

Стоит отметить, что функция записи необходимой информации о пикселе вызывалась полиморфно, что позволяло легко изменять, какие данные сохранялись бы в результате рендеринга. Однако на последних этапах работы она использовалась, в основном, только для сохранения информации о видимых в данный момент метках (с поддержкой глобальной идентификации), а также для составления массива глубин до препятствий (для визуализации того, что видит камера).

3.3. Симуляция rgbd-камеры

Реализованный компонент, производящий симуляцию обычной камеры, позволил также просимулировать работу rgbd-камеры. Для этого достаточно было по полученной от рендерера карте глубин восстановить положение точек в пространстве, а затем занести на карту соответствующее им множество вокселей.

Таким образом, появилась возможность проверить работоспособность существующего подхода к навигации при использовании rgbd-камеры вместо используемого до этого лидара (рис. 2, табл. 1):

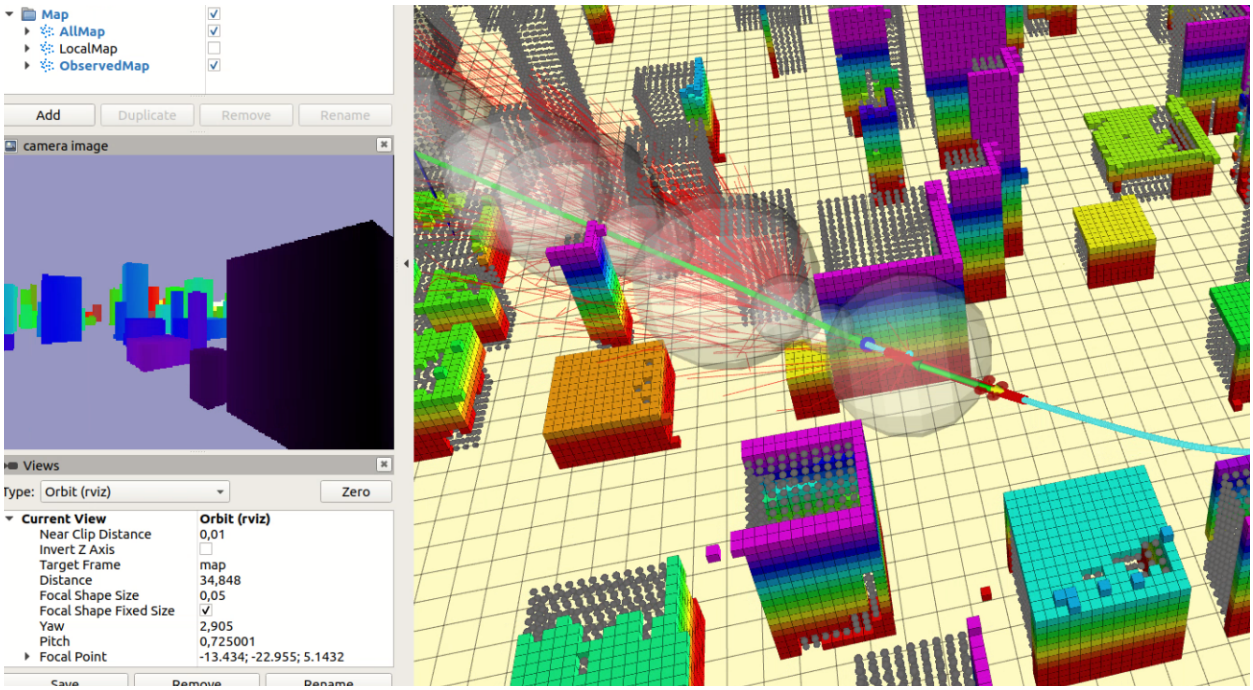

Рис. 2: Симуляция rgbd-камеры

Элемент	Описание
Серые шарики	Точки карты, замощающие полигоны
Разноцветные кубики	Воксели увиденных камерой точек
Серые сферы	Безопасный коридор
Красные отрезки	Дерево возможных маршрутов
Синяя точка	Граница ближайшего участка траектории
Картинка слева	Вывод полученной карты глубин

Таблица 1: Комментарии к элементам изображения

Численные характеристики проделанных тестов приведены в 7.

4. Отслеживание пересечений

Рассмотрим более подробно проблему недостаточно точного восстановления положения точек в пространстве.

Для того чтобы восстановить карту окружающего мира по нескольким снятым изображениям с одной камеры нужно найти на них пары локально похожих пикселей, и для каждой такой пары отметить точку в пространстве, которая предположительно и спроецировалась в них. Однако формально этой паре пикселей соответствует целое множество точек, являющееся пересечением двух пирамид, проходящих через стороны этих пикселей. И полученные таким образом области пространства могут быть очень протяжёнными, если угол между осями этих пирамид будет достаточно мал. Более того, при поступательном движении пирамиды для некоторых пикселей будут практически полностью вкладываться друг в друга. Именно из-за этого и может возникать большая погрешность при аппроксимации положения точек (рис. 3):

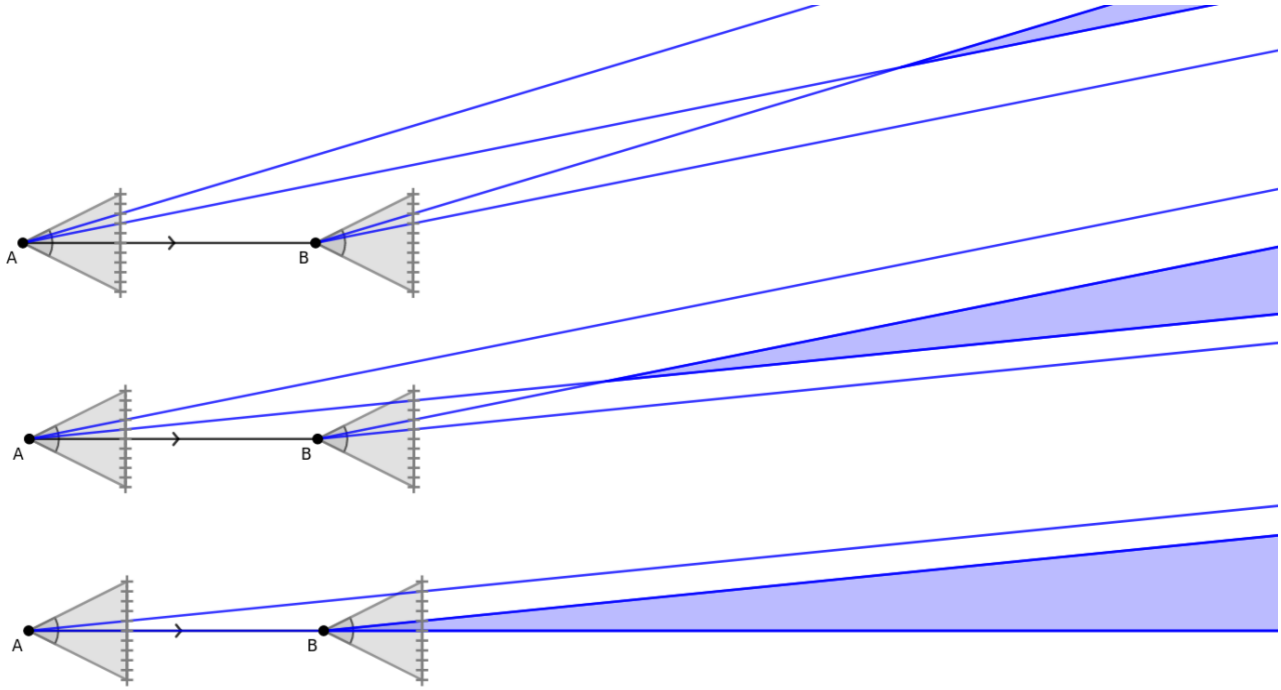

Рис. 3: Иллюстрация проблемы вложения пирамид

На рисунке изображены примеры пересечения углов (пространство заменено на плоскость для простоты), соответствующих трём разным

пикселям. Камеры обозначены серыми треугольниками и для наглядности имеют разрешение в 10 пикселей.

Таким образом, для отслеживания того, каким областям могут принадлежать те или иные точки пространства, необходимо в каком-то виде сохранять информацию о множествах, получаемых при пересечении пирамид, соответствующих одной и той же точке на разных снимках.

Разберём несколько возможных подходов.

4.1. "Честное" пересечение пирамид

Множество, наиболее точным с математической точки зрения образом представляющее возможные положения конкретной точки, очевидно, является многогранником (причислим сюда случаи и с неограниченной фигурой), образованным пересечением всех пирамид для этой точки. Однако расходы на организацию вычислений в данном случае становятся слишком большими по сравнению с пользой от достигаемой точности.

Это происходит из-за того, что пересечением пирамид можно с произвольной необходимой точностью приблизить любую выпуклую фигуру в пространстве. Таким образом, потенциальное число вершин полученного в пересечении многогранника может быть сколь угодно большим. Причём рост числа составляющих его элементов также может происходить довольно быстрым образом: например, в виде добавления нескольких граней за каждое пересечение с новой пирамидой. Из-за этого время и память на пересчёт и хранение каждой такой области постепенно бы увеличивалось в процессе навигации, что могло бы привести, в конечном счёте, к исчерпанию отводимого на вычисления времени.

4.2. Пересечение пирамид без наклона

Оказывается, что теоретической нетривиальности получаемой в пересечении фигуры возможно избежать, накладывая дополнительные ограничения на вид входных данных. Конкретнее, если запретить камере быть наклонённой вокруг своей оси (иметь крен), то получаемые

с её изображений пирамиды также не будут иметь соответствующего наклона. Благодаря этому решение задачи пересечения двух пирамид становится возможным свести к двум задачам по пересечению двух углов на плоскости. Действительно, если спроецировать обе пирамиды на горизонтальную плоскость, то в ней они будут соответствовать двум углам, поскольку эти пирамиды не имели наклона. Также они будут проецироваться в углы и для вертикальной плоскости, проходящей через ось одной из пирамид. После же решения двух задач по пересечению пары углов получится две фигуры (с количеством углов не больше четырёх), по которым возможно восстановить уже само пересечение этих пирамид в пространстве.

В итоге, данный способ позволяет существенно уменьшить объем вычислений в случае пересечения всего двух пирамид, однако накладывает дополнительные ограничения на входные данные, а также не предоставляет естественного способа к хранению и пересчёту информации для ситуации с большим количеством пирамид.

Важно заметить, что отслеживание математически строгого пересечения пирамид само по себе не является осмысленным на практике, поскольку из-за возникающих в действительности погрешностей, например, при определении положения камер, полученные пирамиды могут вообще не пересекаться.

И хоть данный метод и обладал критичным набором недостатков, его концепция оказалась очень важной для дальнейшего хода работы.

4.3. Пересечение в виде усечённого конуса

Развитие идеи в максимальном упрощении решаемой задачи привело к подходу, уже используемому в данной работе. В нём, из-за того что области возможного положения точек получались преимущественно длинными, но тонкими, было решено хранить их в виде надмножеств математически строгого пересечения пирамид, представленных усечёнными прямыми круговыми конусами. Такой способ обеспечивал константный объём данных, сохраняемых для каждой области, поскольку

для одного усечённого конуса достаточно хранить только его вершину, направление оси, а также два расстояния от вершины до соответствующих сечений. Далее, при необходимости обновления данных, пирамида, полученная с нового кадра, также заменялась на содержащий её конус, имеющий ту же вершину и ось, а решение задачи по пересечению усечённого конуса с обычным сводилось их проецированием на плоскость к задаче по пересечению угла и четырёхугольника. В качестве результата пересечения сохранялся усечённый конус с той же вершиной и направлением оси, которые у него были до этого, либо с полученными от нового конуса. Глубины же вычислялись так, чтобы полученный усечённый конус гарантированно содержал пересечение обеих фигур. Таким образом удалось получить также константное время пересчёта для каждой области.

В результате, этот подход позволял свести к минимуму объём вычислений и сохраняемых данных для каждой области, а также не требовал ограничения на наклон камеры. Главным же его недостатком по сравнению с предыдущими методами было то, что в нём информация сохранялась уже о некотором надмножестве точной области. Однако, как выяснилось в процессе тестов, точность локализации точек снижалась несущественным образом по сравнению с преимуществами от простоты их обработки. К примеру, так (рис. 4) соотносятся области, полученные данным способом и с помощью метода из 4.2 для двух снимков: голубыми отрезками обозначены усечённые конусы, синими отрезками (увеличенной толщины для наглядности) изображены диагонали точных пересечений. Несложно заметить, что голубые отрезки, действительно, оказываются ненамного длиннее.

Имеет смысл также отметить, что операция предварительного проецирования обеих фигур в одну плоскость перед непосредственным пересечением оказывается также полезной для устойчивости вычислений при реализации данного подхода на реальном квадрокоптере, поскольку позволяет иногда автоматически обрабатывать ситуации, когда две пирамиды для одной и той же точки могут не пересечься из-за погрешностей при снятии снимков.

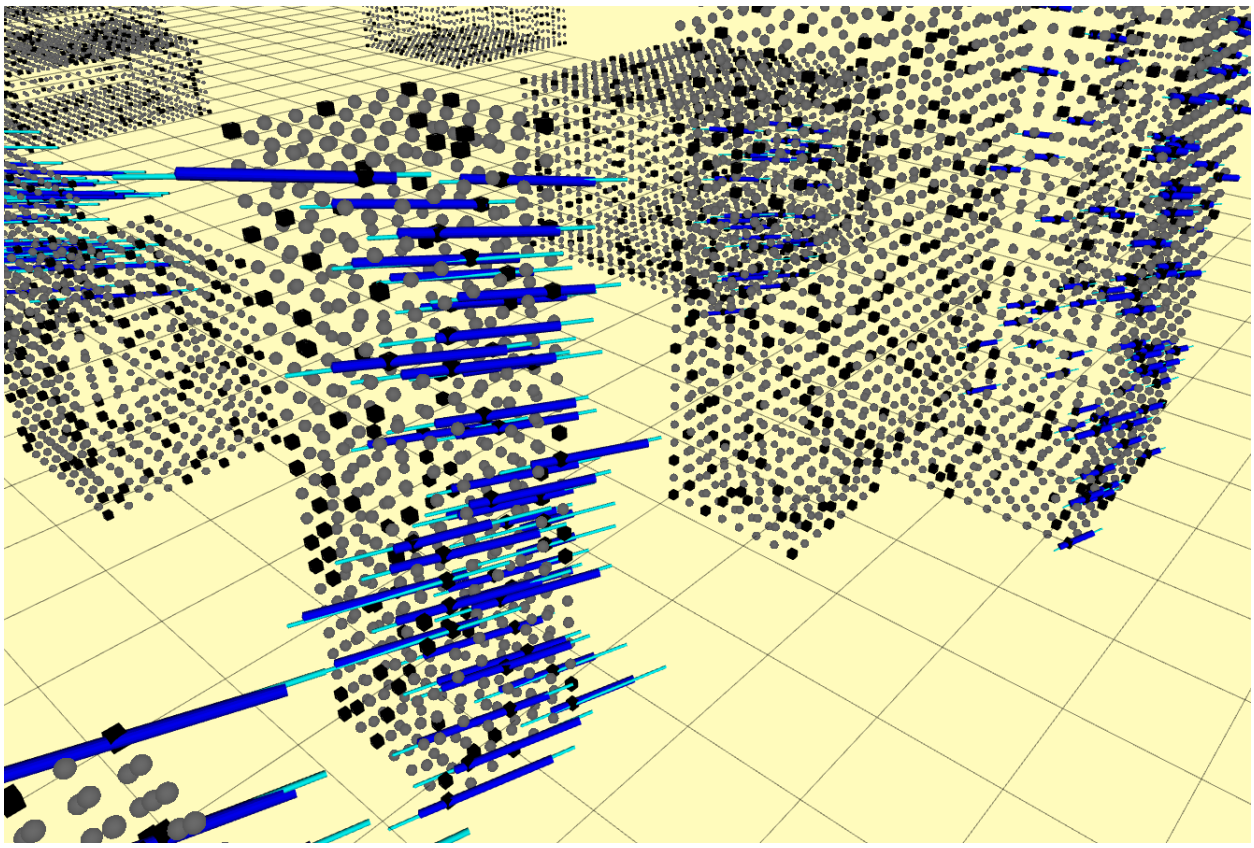

Рис. 4: Соотношение получаемых областей для двух описанных методов

4.4. Доступные эвристики

Данный подход также позволял производить различные эвристики. Например, особенно полезной оказалась эвристика для оценки минимального расстояния до впервые увиденной точки. Её суть заключалась в том, что если первый раз увиденная точка сразу оказалась в центре изображения, то она скорее всего перестала загораживаться каким-то другим препятствием. Поэтому расстояние до неё с предыдущего ракурса должно быть не меньше расстояния до ближайшего из окружающих её препятствий (считая также с предыдущего ракурса).

Благодаря тому, что каждый усечённый конус, соответствующий конкретной точке, своей вершиной имел один из ракурсов съёмки, это позволило применить эвристику, позволяющую обосновывать отсутствие точек препятствий в некоторых областях пространства. Она основывалась на том рассуждении, что если какая-то точка была замечена на двух кадрах, то участки полученных пирамид, соответствующих

пространству между камерой и областью возможного положения точки, обязаны быть пустыми (иначе она была бы не видна).

Заметим, что в обычных методах реконструкции карты информация об увиденных точках используется в основном для указания того, в каких участках пространства могут быть препятствия. Здесь же удаётся получить некоторого рода гарантию, что какая-то область наоборот точно пуста.

Данная эвристика была применена для контроля безопасности ближайшего участка траектории. В сгенерированной траектории выделялся небольшой участок, который затем дискретизировался с помощью воксельной сетки. В результате чего получался набор точек достаточно маленького количества, чтобы можно было отслеживать их попадание в полученные безопасные области. Также для быстрой обработки только что обновлённой траектории было произведено сохранение некоторого числа последних полученных карт глубин (уже для безопасных расстояний).

Рассмотрим пример, в котором полученная с помощью этой эвристики информация может быть действительно полезной. Предположим, что маршрут работа содержит поворот за угол какого-то объекта. Если окажется, что на объекте не найдётся ключевых точек вблизи траектории, то данная эвристика поможет заметить, что какой-то участок пути ещё не был увиден, а значит перед ним есть препятствие, чего нельзя будет сказать с помощью стандартного подхода.

5. Изменение траектории

Теперь, используя представление областей возможного положения точек пространства в виде усечённых конусов, становится более понятным, почему поперечные сдвиги в траектории будут способствовать уточнению локализации точек: оси пересекаемых конусов будут иметь сравнительно большой угол, из-за чего протяжённость каждой области будет быстро убывать.

5.1. Причины для совершения сдвига

Необходимость совершения поперечного сдвига возникает, очевидно, когда точность позиционирования точек оказывается недостаточной для обеспечения безопасности. Этот случай можно отследить, контролируя расстояние до ближайших концов всех сохранённых на карте усечённых конусов.

Также довольно важным является отслеживание ситуаций, когда некоторая группа точек вблизи траектории просто обладает большой неопределённостью положения. Это помогло бы обрабатывать случаи, когда из-за искажения реконструируемой карты может быть невозможным проложить маршрут в пустом на самом деле месте. Для реализации этой функции, в конечном счёте, потребовалось учитывать протяжённость, ориентацию оси и расстояние до усечённых конусов, соответствующих обозреваемым в данный момент ключевым точкам. Эта информация также позволяла определить, в каком направлении лучше всего производить сдвиг, а также какой он должен быть величины.

5.2. Форма сдвига

Как упоминалось ранее, генерируемая траектория представлялась в виде последовательности кривых Безье. Поскольку каждая кривая соответствовала своей сфере полётного коридора, было решено добавлять сдвиги в траекторию посредством изменения формы одной из кривых. Это также позволяло автоматически обеспечить безопасность нового

участка при условии, что его опорные точки находятся внутри той же сферы.

Для всех сдвигов был выбран общий шаблон в виде спирали (рис. 5):

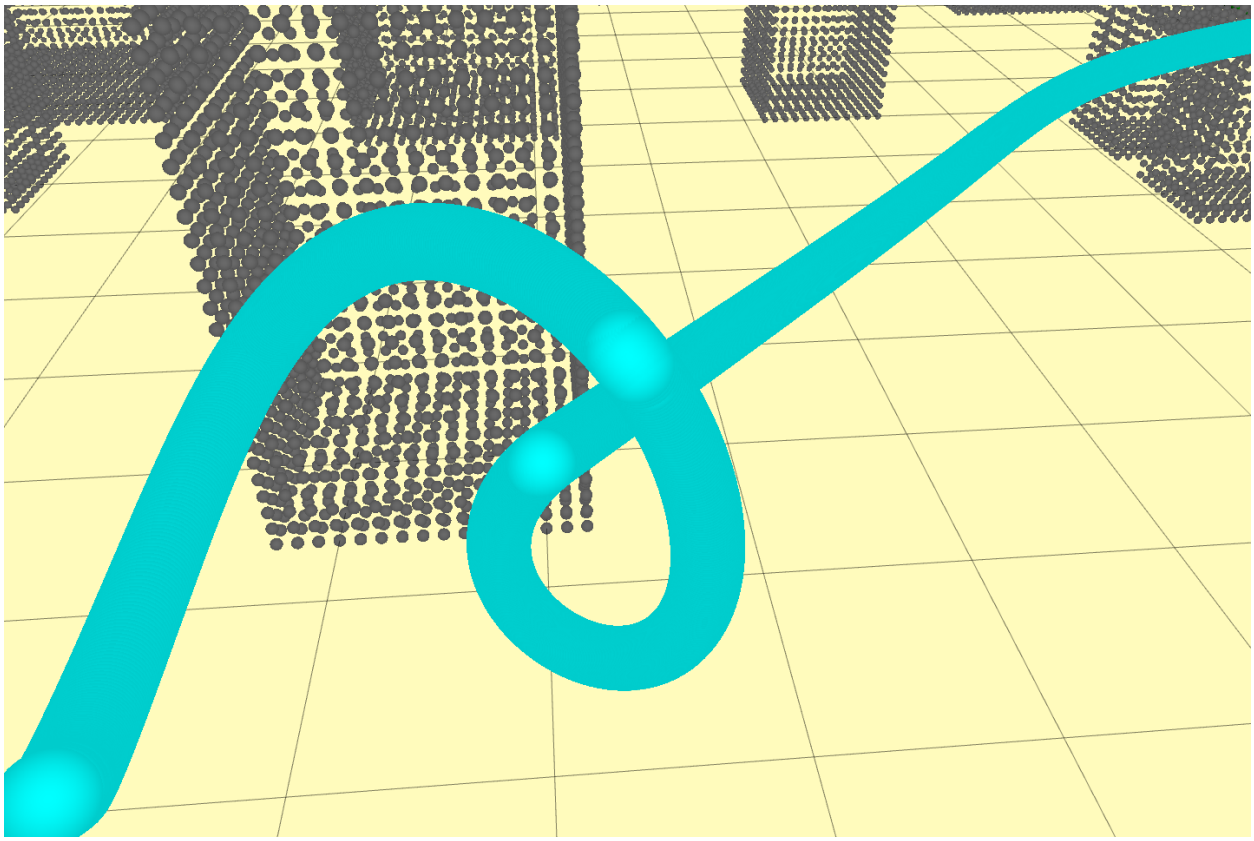

Рис. 5: Пример сдвига, произведённого на старте

Это позволяло, во-первых, сохранить гладкость траектории, а во-вторых, такая форма организовывала смещение по разным направлениям, благодаря чему получалось бы уточнить положение потенциально большего числа точек. Тем не менее, основное направление сдвига и его величина рассчитывались для каждого сдвига отдельно, как было написано в 5.1. Также, при необходимости, форму можно было сделать более плоской, уменьшив число используемых опорных точек.

5.3. Детали реализации

Компонент, выполняющий вставку сдвигов было решено реализовать в виде отдельного узла, производящего постобработку сгенерированной траектории. Это позволяло не изменять существенным образом

ход работы основного алгоритма навигации. Однако для этого новому узлу потребовалось также пересылать дополнительные данные (помимо самой траектории), содержащие необходимую информацию о полётном коридоре.

5.4. Повороты во время полёта

Как отмечалось в 4.4, в процессе полёта также производился контроль безопасности ближайшего участка траектории. При наступлении момента, когда для всего следующего интервала заданной длины появлялась гарантия его безопасности, узел постобработки также получал возможность поворачивать камеру квадрокоптера во время полёта вдоль этого интервала. Благодаря этому добавлялся ещё один способ уточнения локализации точек, так как объекты, находящиеся по бокам от траектории автоматически имели бы большие смещения на разных кадрах, и главной проблемой была необходимость "посмотреть" именно в их сторону. Направление для поворота рассчитывалось тем же способом, что и направление для сдвига.

6. Изменения в структуре системы

6.1. Начальная структура

До начала работы структура существующей системы выглядела следующим образом:

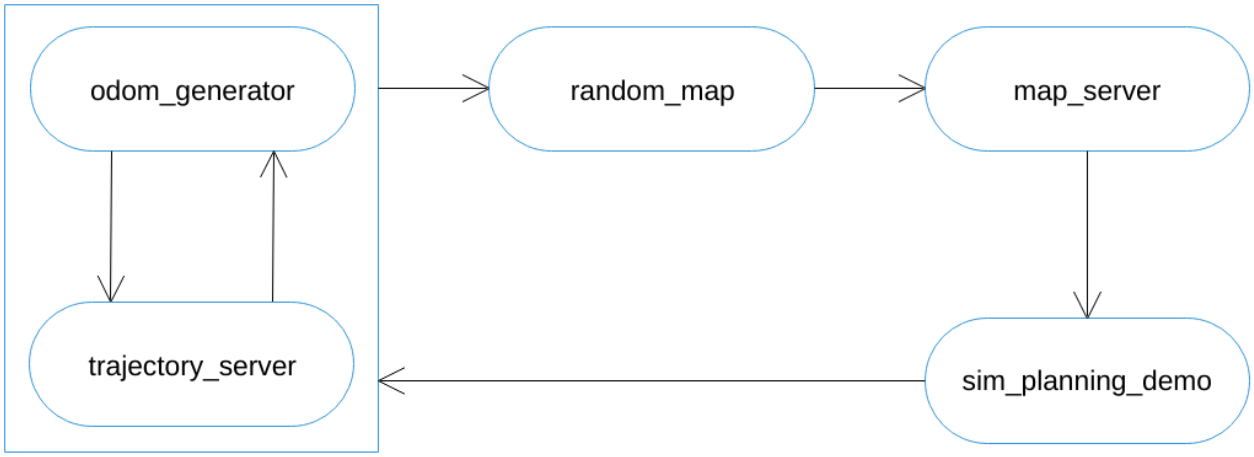

Рис. 6: Изначальная структура системы

Сразу отметим, что схемы в данном разделе изображают только самые важные для понимания работы системы компоненты и связи между ними.

Работа алгоритма навигации происходила в узле sim_planning_demo. Сгенерированная траектория передавалась узлу trajectory_server. Этот узел на рисунке сгруппирован с узлом odom_generator, поскольку они оба отвечали за распространение текущих координат квадрокоптера, работая в паре: узел trajectory_server вычислял текущее положение, основываясь на времени, прошедшем от момента получения траектории; узел же odom_generator обеспечивал своевременный запуск этих вычислений для обновления положения робота, а так же производил конвертацию сообщения с новым положением в необходимый некоторым другим узлам формат.

Узлы random_map и map_server отвечали за генерацию карты и симуляцию лидара соответственно.

6.2. Конечная структура

К концу работы структура системы частично изменилась (рис. 7):

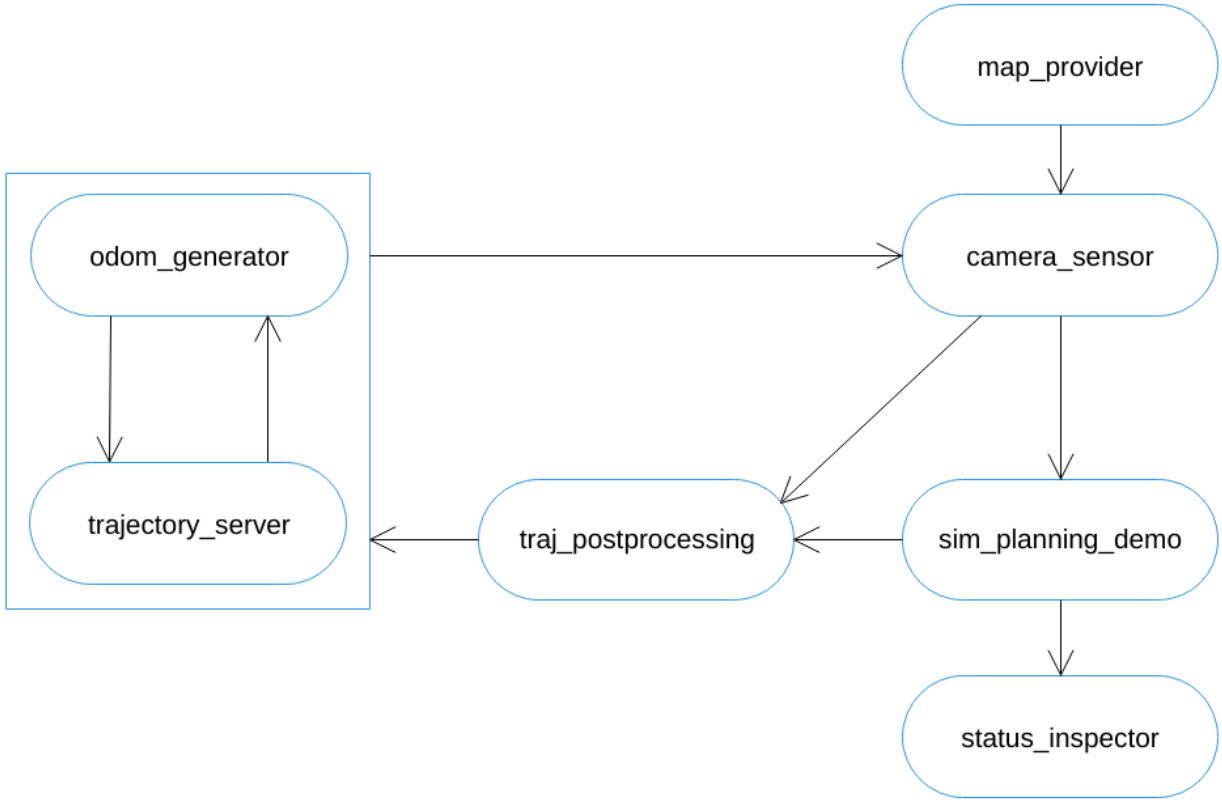

Рис. 7: Конечная структура системы

Узел random_map был заменён на узел map_provider, производящий генерацию карты уже с новым представлением. Также был исправлен архитектурный недочёт, заключающийся в том, что узел random_map, не обязан был находится внутри образованного "цикла", так как составление карты не требовало, на самом деле, никакой информации от других узлов. Это также позволило останавливать работу узла map_provider, после того как он передавал другим узлам требуемые данные о карте.

Существенные изменения коснулись также узла map_server, который теперь, в зависимости от конфигурации запуска позволял симулировать лидар (возвращающий все точки в некотором радиусе), rgbd-камеру, а также обычную камеру (с отслеживанием областей возможного положения точек или без него). В этом же узле, который получил название camera_sensor, производится также расчёт возможных направлений для сдвигов или поворотов камеры.

Узел traj_postprocessing, соответственно, производил постобработку сгенерированной в узле sim_planning_demo траектории.

Также для возможности автоматически запустить навигацию несколько раз с контролем произошедших сбоев был добавлен узел status_inspector, следящий за возникающими ошибками.

7. Анализ результатов

Проверка работоспособности разработанного подхода была проведена с помощью группы тестов, позволяющей сравнить качество навигации с использованием различных датчиков (рис. 8). Для этого при одинаковых по возможности параметрах (включая как разрешение камеры, так и саму карту) было произведено по 300 запусков навигации с отслеживанием возникающих во время полёта сбоев. "Уникальные" параметры возникали, в основном, для случая с камеры без использования сдвигов, поскольку в этом случае требовалось просимулировать работу обычного способа восстановления точек, в котором, например, для этого используется окно в некоторое число кадров, причём нигде в другом месте это количество не фигурировало.

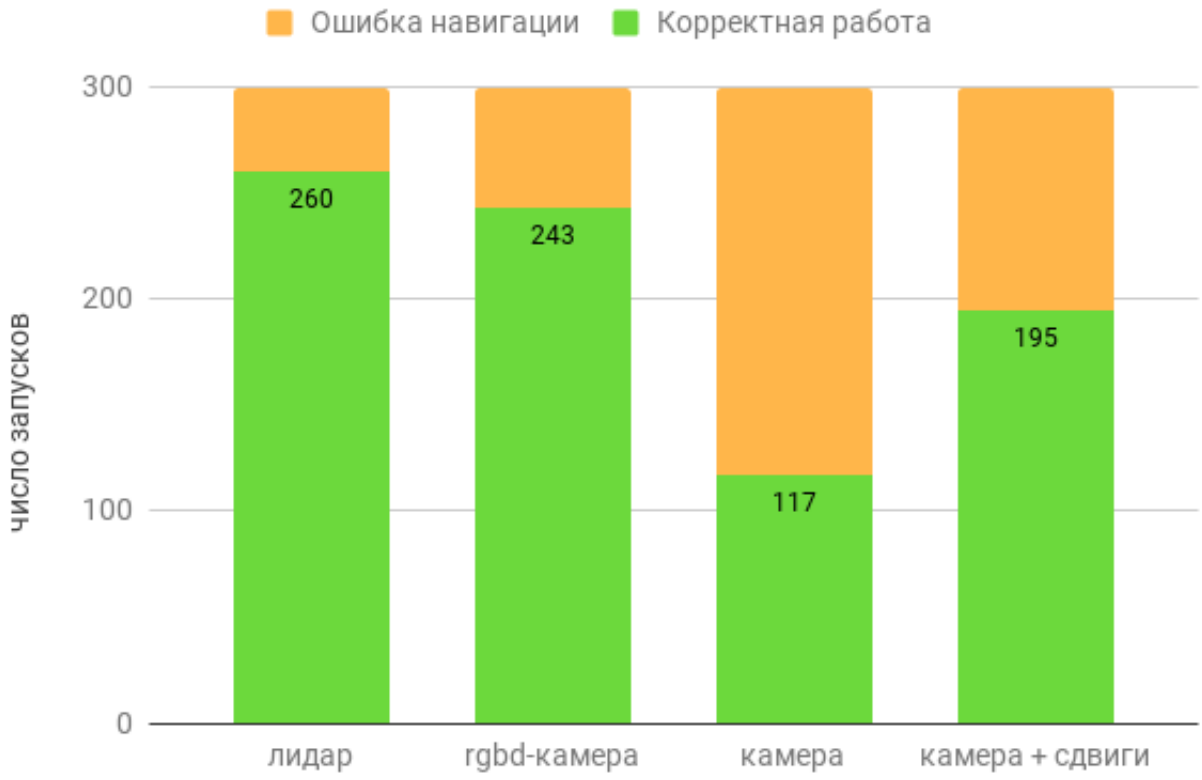

Рис. 8: Сравнение качества навигации при разных датчиках

Как можно убедиться, алгоритм навигации, в целом, остаётся работоспособным и при использовании rgbd-камеры вместо лидара. Случай

с камерой без использования сдвигов приводит к существенно большему числу ошибок, чем случай со сдвигами.

Важно отметить, что возникновение любой проблемы в алгоритме навигации считалось сбоем, после чего полёт немедленно останавливался. А подавляющее большинство сбоев происходило из-за ошибок во время поиска безопасного коридора (с чем, в основном, и связаны плохие результаты у обычной камеры: из-за плохой локализации точек в какой-то момент могло быть невозможно проложить дальнейший маршрут). Поэтому, в действительности, если бы тесты производились на реальном квадрокоптере, почти во всех случаях ошибок навигации робот скорее всего просто не смог бы дальше продолжать полёт. Принудительное продолжение полёта после обнаружения сбоя было решено не реализовывать, поскольку это бы потребовало внесения существенных изменений в основной алгоритм навигации.

Заметим также, что предлагаемый подход не должен привести к заметному увеличению вычислительных затрат, поскольку, как отмечалось в 4.3, пересчёт каждой области сводится к задаче с фиксированным временем решения, и поэтому общее время на пересчёт областей для каждого нового снимка камеры будет линейно зависеть от числа распознанных ключевых точек.

Заключение

Таким образом, при выполнении данной работы были получены следующие результаты:

- изучена предметная область автономной навигации квадрокоптеров, сделан выбор существующей реализации навигации (в симуляции) для взятия за основу,
- внесены необходимые изменения в способ представления карты, существенно расширен набор доступных сенсоров для симуляции,
- подтверждена работоспособность существующего подхода при использовании rgbd-камеры,
- разработан и реализован метод отслеживания областей возможного положения точек, основанный на представлении областей в виде усечённых конусов,
- реализована идея с поперечными сдвигами,
- проверена работоспособность разработанного подхода.

Список литературы

[1] AliExpress. American 3D 16 line LiDAR PUCK VLP-16 lidar sensor. — URL: https://aliexpress.ru/item/4000206764459.html (дата обращения: 25.05.2020).

[2] Library Wiley Online. Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments. — URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21842 (дата обращения: 13.12.2019).

[3] Lidar Velodyne. Puck. — URL: https://velodynelidar.com/products/puck/ (дата обращения: 25.05.2020).

[4] Lidar Velodyne. Puck LITE. — URL: https://velodynelidar.com/products/puck-lite/ (дата обращения: 25.05.2020).

[5] RealSense Intel. Intel RealSense Depth Camera D435i. — URL: https://store.intelrealsense.com/buy-intel-realsense-depth-camera-d435i.html (дата обращения: 25.05.2020).

[6] Skydio. Skydio 2. — URL: https://www.skydio.com/ (дата обращения: 19.05.2020).

[7] University Carnegie Mellon. P-CAP: Pre-computed Alternative Paths to Enable Aggressive Aerial Maneuvers in Cluttered Environments. — URL: https://www.ri.cmu.edu/publications/p-cap-pre-computed-alternative-paths-to-enable-aggressive-\aerial-maneuvers-in-cluttered-environments/ (дата обращения: 13.12.2019).