Глава 1

Постановка задачи

Жизнь в 21 веке тесно связана с технологиями. Каждый второй человек сейчас имеет полную по Тьюрингу вычислительную машину, которую называют телефоном. Все эти технологические новшества помогают сделать нашу и без того комфортную жизнь, еще более комфортной. Следующей вехой развития я вижу развитие IoT (интернет - вещей).

1.1 IoT

Термин IoT, или Интернет вещей, относится к коллективной сети подключенных устройств и технологии, которая облегчает связь между устройствами и облаком, а также между самими устройствами. Благодаря появлению недорогих компьютерных микросхем и телекоммуникаций с высокой пропускной способностью у нас теперь есть миллиарды устройств, подключенных к Интернету. Это означает, что повседневные устройства, такие как зубные щетки, пылесосы, автомобили и механические установки, могут использовать датчики для сбора данных и разумного реагирования на действия пользователей.

1.2 Cloud robotics

В связи с развитием технологий передачи информации на большие растояния с большой скоростью, многие робототехнические компании заинтересовались IoT и тем, как можно скрестить их с робототехникой в целом. Такой симбиоз получился очень удачным и открыл новую область робототехники - cloud robotics (облачная робототехника).

1.3 Цель работы

Целью мое работы будет разработка ПО для взаимодействия робота с некоторой облачной инфраструктурой, для задач SLAM алгоритма. То есть задача SLAM будет решаться в облаке.

Возникает резонный вопрос, зачем переносить SLAM в облако? Вот несколько ключевых причин:

- Снижение загруженности CPU/GPU на бортовом компьютере робота
- Снижение зависимости робота от локально построенной карты помещения
- Снижение потребления заряда батареи
- Снижение стоимости робота

Глава 2

Обзор состояния науки и техники

Для начала выделим необходимые технология для обеспечения работоспособности Cloud SLAM:

- Стабильная сеть с высокой пропускной способностью
- Платформы для разработки топологии взаимодействия между компонентами SLAM, которые имеют низкий overhead
- Технологии для хранения данных, которые имеют низкий overhead
- Технологии для передачи потоковых данных, которые имеют низкий overhead

Из этих абстрактных требований мы можем понять, что для обечпечения жизнеспособности такого метода вычисления карты и положения на ней требуются высокий уровень технического обеспечения и знаний. Такие жесткие требования проистекают от требования в низкой задержке. Если SLAM алгоритм имеет высокую задержку, то он оказывается бесполезным для работы.

2.1 Обзор литературы

Мной был проведен поиск литературы по схожей проблеме. Вот несколько статей:

- Supun Kamburugamuve, Leif Christiansen, Geoffrey Fox. - A Framework for Real Time Processing of Sensor Data in Cloud. - Indiana University:April 2015. - 12 p.
- Supun Kamburugamuve, Hegjing He, Geoffrey Fox. - Cloud-based Parallel Implementatio of SLAM for Mobile Robots. - Indiana University:December 2017. - 8 p.

Обе эти статьи описывают типичное решения для задачи SLAM в облаке. В моей работе я хочу провести расширение идей вычислений в облаке для задач картографирования и локализации на карте.

Глава 3

Реализация

Архитектура подобного приложения предполагает 3 слоя:

1. Gateway layer
2. Message Broker layer
3. Processing layer

На Gateway layer мы предпологаем некое приложение или уже готовую технологию для связи драйвера робота и Message Broker - ов, таких как Kafka или RabbitMQ. Типичные решения - отсутсвуют(ну или я не нашел).

На Message Broker layer мы распологаем брокеров сообщении, которые будут выступать в роли очереди для скопившихся сообщений, которые еще не были обработаны на Processing layer. Типичное решение - kafka, RabbitMQ

На Processing layer распологается сама логика. В нашем случае - SLAM. На картинке выше представлен SLAM на основе фильтра частиц, который хорошо параллелиться, чего нельзя сказать о графовых SLAM. Типичное решение - Apache Storm из-за его ориентацию на stream (те на потоки данных), а не на batch (пакеты) данных.

Глава 4

Apache Storm

Для реализации топологии Apache Storm (на 3-ем уровне) есть моменты которые нужно учитывать. В данном фреймворке всем управляет Nimbus, он занимается распределением задач и сериализацией потоков данных в системе. Однако, управление идет не от него напрямую, а черезе специальную ноду - Zookeeper, которая управляет кластером машин (может и виртуальных), на которых запущенна сама топология - Spout (принимающие и передающие tuple-ам информацию) и Bolt (принимающие, обрабатывающие и передающие tuple информацию) нод. Визуально это выглядит следующим образом:

Глава 5

Обзор традиционных решений

Cloud robotics, к сожалению, пока всего лишь тенденция. SLAM прочно закрепился в inter (внутреннем) взаимодействии, к тому же, многие методы SLAM хорошо вписываются в архитектуру CPU, за счет чего, становяться более оптимизированными.

Однако, в традиционном использовании SLAM алгоритмов мы ограничены производительностью бортового компьютера робота, в то время как на сервере, мы можем создать целый кластер компьютеров, которые в десятки раз быстрее буду обрабатывать данные с сенсоров. Хотелось бы сказать, что SLAM алгоритмы, которые используются на сервере и на роботах, по логике работы ничем не отличаются, поэтому ,чтобы выбрать наиболее подходящий нам SLAM алгоритм, мы проведем обзор.

Сравнивать будем по следующим критериям:

1. Используемые датчики, их максимальное количество.
2. Performance построения карты.
3. Performance локализации.
4. Accuracy построения карты.
5. Accuracy локализации.
6. Совместимость с пакетом robot_localization.

Performance построения карты сложно оценить полностью независимо от локализации. Поэтому, за оценку этого критерия будем брать загруженность CPU компьютера со следующими характеристиками:

- CPU: AMD Ryzen 7 4800H 8 kernel 2.9 GHz (4.2 GHz - Turbo)
- Mem: 16 GB DDR4 3200
- GPU: NVIDIA GeForce GTX 1650 Ti 4 Gb video memory

Чтобы оценить performance локализации, будем замерять время от старта локализации, до того момента, пока возможные положения и ориентация робота не предут в стабильное состояние. Те будем замерять время на локализацию. При этом, для чистоты эксперемента, будем локализироваться по одной и той же траектории.

Дабы оценить Accuracy построения карты, будем сравнивать построенную алгоритмом карту и карту помещения построенную в CAD системе.

Accuracy локализации можно замерить встав на некоторую точку с заранее известными координатами. Затем провести сравнение координат, полученных из tf топиков и заранее известных координат.

Так же для этих целей, у меня есть bag файл, любезно записанный моими коллегами. Будем использовать его. Скачать bag файл можно на моем github: `https://github.com/sees1/MagMPEI.git`

5.1 Обзор SLAM

5.1.1 OpenSlam's Gmapping + AMCL

slam_gmapping - первая часть laser-base SLAM, который отвечает за mapping. Предположительно на фильтре частиц. Хорошо поддерживается, много документации, много примеров в открытом доступе.
AMCL - вторая часть laser-base SLAM, которая отвечает за localization. Основан на фильтре частиц. Хорошо поддерживается, много документации, много примеров в открытом доступе.

Характеристики:

Используемые датчики, их максимальное количество: odometry(1 unit) + laser(1 unit)

Performance построение карты: -

Performance локализации: -

Accuracy построение карты: -

Accuracy локализации: -

Совместимость с пакетом RL: совместим

slam_gmapping repo: `https://github.com/ros-perception/slam_gmapping`

AMCL repo: `https://github.com/ros-planning/navigation/tree/noetic-devel/amcl`

5.1.2 hector_slam

Полноценный laser-base SLAM. Предположительно на фильтре частиц. Динамически строит карту (отправляет ее в топик) и динамически локализируется на строящейся карте.

Есть способы пробросить динамическую карту в move_base. Использование локализаций из amcl не нужно, потому что hector при построении, автоматом локализуется на карте.
Хорошо поддерживается.

Характеристики:

Используемые датчики, их максимальное количество: laser(1 unit) + Optional:odometry(1 unit)

Performance построение карты: -

Performance локализации: -

Accuracy построение карты: -

Accuracy локализации: -

Совместимость с пакетом RL: совместим

repo:`https://github.com/tu-darmstadt-ros-pkg/hector_slam`

5.1.3 rtabmap_ros

SLAM работающий почти со всеми типами sensor_msgs. Основан на графах. Хорошо поддерживается, много документации.
Можно использовать, как самостоятельно решение, без move_base пакета, для планирования, т.к. есть еще и Global/Local Planner.

Характеристики:

Используемые датчики, их максимальное количество: laser(1 unit)/lidar(1 unit)/camera(1 unit) + Optional:odometry(1 unit) + Optional:IMU(1 unit) + Optional:GPS(1 unit)

Performance построение карты: -

Performance локализации: -

Accuracy построение карты: -

Accuracy локализации: -

Совместимость с пакетом RL: совместим

repo:`https://github.com/introlab/rtabmap_ros`

5.1.4 cartographer_ros

SLAM работающий в основном с laserScan/PointCloud. Основан на графах. Есть возможность использовать multiSLAM. Хорошо поддерживается, много документации.
Можно использовать, как самостоятельно решение, без move_base пакета, для планирования, т.к. есть еще и Global/Local Planner.

Характеристики:

Используемые датчики, их максимальное количество: laser(1 unit)/lidar(1 unit)/camera(1 unit) + Optional:odometry(1 unit) + Optional:IMU(1 unit) + Optional:GPS(1 unit)

Performance построение карты: -
Performance локализации: -
Accuracy построение карты: -
Accuracy локализации: -
Совместимость с пакетом RL: совместим
repo:`https://github.com/cartographer-project/cartographer_ros`

5.1.5 slam_karto

SLAM работающий с laserScan. Предположительно основан на графах. Последний коммит 2020 года

Характеристики:

Используемые датчики, их максимальное количество: laser(1 unit) + Optional:odometry(1 unit)
Performance построение карты: -
Performance локализации: -
Accuracy построение карты: -
Accuracy локализации: -
Совместимость с пакетом RL: совместим
repo:`https://github.com/ros-perception/slam_karto`

5.1.6 mrpt_slam

Обертка над SLAM алгоритмами из библиотеки Mobile Robot Programming Toolkit (MRPT). Есть как EKF алгоритмы SLAM, так и RBPF SLAM, Graph SLAM. Последний коммит 2023 года (хорошо поддерживается)

Характеристики:

Используемые датчики, их максимальное количество: Probably:laser(1 unit)/lidar(1 unit) + Optional:odometry(1 unit)
Performance построение карты: -
Performance локализации: -
Accuracy построение карты: -
Accuracy локализации: -
Совместимость с пакетом RL: совместим
repo:`https://github.com/mrpt-ros-pkg/mrpt_slam`

5.1.7 rgbd_slam

SLAM работающий с rgbd камерами, картинка с которых преобразуется в плотное облако точек и используется в качестве карты. Предположительно основан на фильтре цастиц. Последний коммит 2018 года.

Характеристики:

Используемые датчики, их максимальное количество: Probably:lidar(1 unit) or rgbdCam(1 unit) + odometry(1 unit)
Performance построение карты: -
Performance локализации: -
Accuracy построение карты: -
Accuracy локализации: -
Совместимость с пакетом RL: совместим
repo:`https://github.com/felixendres/rgbdslam_v2`

5.1.8 ohm_tsd_slam

SLAM работающий с laserScan. Основан на TSDF (truncated signed distance transform) алгоритме. Последний коммит 2020 года.

Характеристики:
Используемые датчики, их максимальное количество: laser(1 unit) + Optional:odometry(1 unit)
Performance построение карты: -
Performance локализации: -
Accuracy построение карты: -
Accuracy локализации: -
Совместимость с пакетом RL: совместим
repo:`https://github.com/autonohm/ohm_tsd_slam`

5.1.9 tiny_slam_ros_cpp

SLAM работающий с laserScan. Предположительно основан на фильтре частиц. Последний коммит 2017 года.

Характеристики:
Используемые датчики, их максимальное количество: laser(1 unit) + Optional:odometry(1 unit)
Performance построение карты: -
Performance локализации: -
Accuracy построение карты: -
Accuracy локализации: -
Совместимость с пакетом RL: совместим
repo:`https://github.com/OSLL/tiny-slam-ros-cpp`

5.1.10 stereo_slam

SLAM работающий со stereo cam. Предположительно основан на графах. Последний коммит 2017 года.

Характеристики:

Используемые датчики, их максимальное количество: stereo camera(1 unit) + Optional:odometry(1 unit)
Performance построение карты: -
Performance локализации: -
Accuracy построение карты: -
Accuracy локализации: -
Совместимость с пакетом RL: совместим
repo:`https://github.com/srv/stereo_slam`

5.1.11 lsd_slam

SLAM работающий со mono cam. Предположительно основан на графах. Последний коммит 2014 года.

Характеристики:
Используемые датчики, их максимальное количество: mono camera(1 unit) + Optional:odometry(1 unit)
Performance построение карты: -
Performance локализации: -
Accuracy построение карты: -
Accuracy локализации: -
Совместимость с пакетом RL: совместим
repo:`https://github.com/tum-vision/lsd_slam`

5.1.12 crsm_slam

SLAM работающий с laserScan. Предположительно основан на фильтре частиц. Последний коммит 2016 года.

Характеристики:
Используемые датчики, их максимальное количество: laser(1 unit) + Optional:odometry(1 unit)
Performance построение карты: -
Performance локализации: -
Accuracy построение карты: -
Accuracy локализации: -
Совместимость с пакетом RL: совместим
repo:`https://github.com/etsardou/crsm-slam-ros-pkg/tree/indigo-devel`

5.1.13 vslam

Очень мало информации.
repo:`https://code.ros.org/svn/ros-pkg/stacks/vslam/trunk`