Санкт-Петербургский государственный университет

Кафедра системного программирования
Программная инженерия

Милосердова Любовь Михайловна

Создание pip-пакета для C++ библиотеки mrob

Отчёт по учебной практике

Научный руководитель:
ст. преп. Я. А. Кириленко

Консультант:
Инженер-исследователь, Сколтех А. В. Корнилова

Санкт-Петербург
2021

Оглавление

Введение

Mobile Robotics library (mrob) [4] — это библиотека лаборатории мобильной робототехники Сколковского института науки и технологий. Библиотека используется для реализации исследовательских и промышленных проектов в области навигации в робототехнике, а также для обучения студентов магистратуры. Она включает в себя набор функций, полезных для планирования пути и его оптимизации, а также вспомогательные функции, такие как геометрические преобразования в трехмерном пространстве с использованием групп Ли $SO(3)$ и $SE(3)$, построение фактор-графов для SLAM задач (Simultaneous localization and mapping) и многое другое.

Исходный код данной библиотеки написан на языке C++. Однако для прототипирования проектов, постановки экспериментов и обучения студентов традиционно используется Python, как самый популярный язык программирования [17], в том числе среди инженеров анализа данных [18].

Использование библиотеки mrob в коде на языке Python возможно благодаря библиотеке pybind [5], предоставляющей возможность сопоставлять типы данных, стуруктуры данных, перечисления, итераторы, функции, типы исключений и многое другое из кода, написанного на языке C++, в код на языке Python.

Поскольку библиотека распространяется в исходных кодах, пользователю необходимо самостоятельно компилировать её в правильном окружении для конкретной версии Python и операционной системы, что неудобно и занимает определенное время. Хотелось бы упростить этот процесс.

Библиотеки Python распространяются в виде пакетов. Под пакетом принято понимать файл определенного формата, который можно скачать и установить. Таким образом, задача состоит в том, чтобы создать пакет для библиотеки mrob, который будет доступен на широком спектре платформ.

1. Обзор предметной области

Pip — это рекомендованный менеджер Python пакетов. Pip позволяет устанавливать пакеты как локально или из основного индекса пакетов Python, PyPI [12], так и из альтернативных источников, например, из специального тренировочного индекса Test PyPI [13] или из репозиториев на GitHub.

Индекс пакетов Python (PyPI) — это хранилище программного обеспечения для языка Python. Разработчики используют PyPI для распространения своего программного обеспечения. После загрузки пакета в PyPI пользователи могут найти и установить его с помощью простой команды `pip install “SomeProject“`.

1.1. Устройство pip-пакетов

Существует два формата распространения:

Source Distribution (sdist) — формат распространения, который предоставляет метаданные и основные исходные файлы, необходимые для сборки и установки.

Wheel (колесо) — это формат распространения, который обеспечивает более быструю установку по сравнению с Source Distribution за счет того, что он предварительно собран.
Название файла колеса определяет подходит ли данное колесо для установки на конкретной машине и выглядит так:

packageName—packageVersion—pythonVersion—operatingSystem—processorArchitecture.whl

Pip может установить пакет либо из sdist, либо из колёс, но если оба они присутствуют в PyPI, pip предпочтет колесо. Если pip не найдет подходящее колесо для установки, он построит колесо локально из sdist.

Для того, чтобы была возможность собрать колесо из проекта, в нем должны находиться следующие файлы [8].

- __init__.py

  - определяет к каким классам пользователь может получить доступ через интерфейс пакета;
  - требуется для импорта каталога в качестве пакета.

- Файлы сценария сборки (например, setup.py) — сообщают информацию о пакете (например, имя и версию), а также указывают все зависимости пакета.

- LICENSE — сообщает пользователям, которые устанавливают пакет, условия, на которых они могут использовать его.

- README.md (опционально) — является развернутым описанием пакета.

1.2. Библиотека mrob

Библиотека mrob имеет несколько модулей: libSE3, libPCRegistration, libFGraph, libcommon и mrob. Основной модуль, mrob, использует pybind11 [5] — библиотеку, позволяющую осуществлять привязку Python версий 2.7 и 3.5+ к существующему коду C++. Данный модуль зависит от версии Python, что связано с тем, что Python гарантирует совместимость ABI лишь в патч(микро) версиях [11]. Модуль mrob линкуется с остальными модулями, которые от версии Python не зависят.

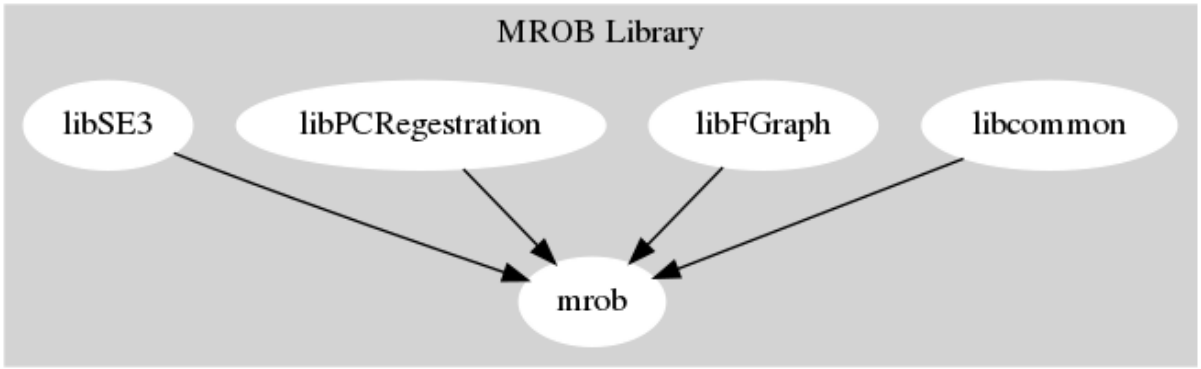

Рис. 1: Модули библиотеки mrob

В проекте для обеспечения непрерывной интеграции и доставки (CI/CD) используется Circle CI [3].

Сборка библиотеки mrob осуществляется с использованием системы сборки CMake.

1.3. Способы поддержания кроссплатформенности

Исследователи и обучающиеся пользуются библиотекой на основных десктопных платформах, таких как GNU/Linux, macOS, Windows. Поэтому необходимо обеспечить кроссплатформенность в первую очередь для данных операционных систем.

1.3.1. GNU/Linux

Семейство операционных систем на базе ядра Linux имеет множество дистрибутивов с различными наборами библиотек и утилит.

Наивной идеей может быть сборка колёс под каждый из дистрибутивов GNU/Linux. Это решение неразумно из-за количества дистрибутивов — в настоящее время их более 600 и около половины из них поддерживаются в актуальном состоянии [6]. Например, хронология жизни одного только семейства дистрибутивов Ubuntu выглядит так:

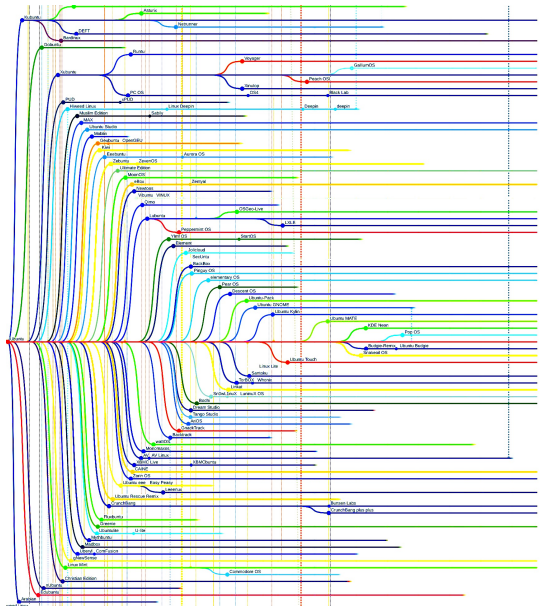

Рис. 2: Семейство дистрибутивов Ubuntu

Другим решением является идея собрать колесо на определенном дистрибутиве, который будет совместим с остальными.

У всех модулей библиотеки есть зависимости от других динамических библиотек, часть из которых предоставляет дистрибутив. Эти динамические библиотеки в свою очередь тоже зависят от каких-то библиотек. В операционной системе GNU/Linux на вершине иерархии зависимостей библиотек находится библиотека glibc, которая совершает системные вызовы, то есть обращается к ядру операционной системы и, таким образом, зависит лишь от него. Сборка, имеющая зависимость от ранней версии glibc, позволит гарантировать, что исполняемый файл, использующий библиотеку mrob, не вызовет какую-либо отсутствующую функцию.

Достаточно ранняя версия библиотеки glibc предоставляется дистрибутивом Centos 6.

1.3.2. macOS

Новая версия операционной системы macOS (ранее называвшейся OS X) выходит раз в год. Согласно найденной статистике[1] [16] на октябрь 2021 года менее 1,5% пользователей операционной системы macOS пользуются версией OS X 10.10 (Yosemite).

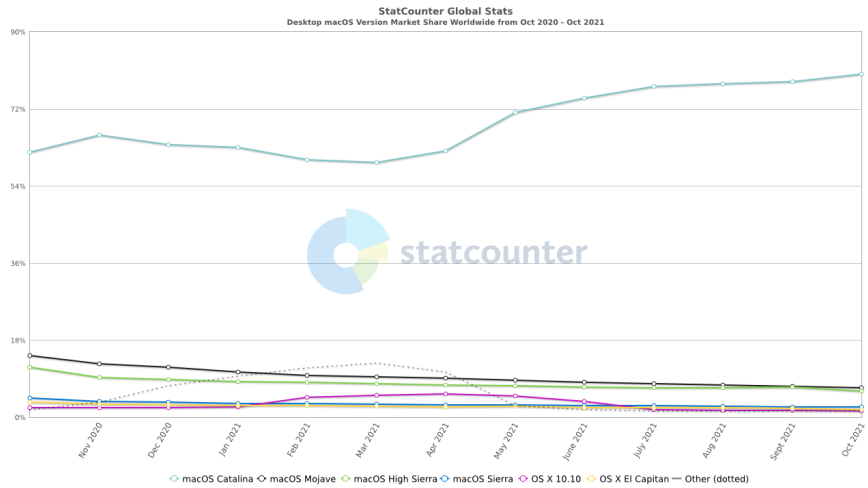

Рис. 3: Статистика использования версий macOS

[1]В статистике отсутствуют версия macOS 11 Big Sur, так как утверждается, что Apple отображает версию macOS Catalin 10.15 вместо macOS Big Sur 11. Подробнее об ошибке можно прочитать здесь.

Несмотря на то что количество версий данной операционной системы значительно меньше количества дистрибутивов GNU/Linux, сборка колёс отдельно для каждой версии macOS нецелесообразна.

XCode — набор инструментов для разработки ПО для платформ Apple. Версии операционной системы отличаются между собой версиями SDK, используемых в XCode. В случае наивного решения можно было бы собирать библиотеку для необходимой версии macOS устанавливая соответствующую версию SDK. Однако минимальную поддерживаемую версию можно указать на уровне опций компилятора, тогда на этой версии операционной системы и на всех последующих библиотека будет доступна. Согласно информации на официальном сайте Apple [1] при сборке на свежих версиях macOS можно добиться совместимости с версиями 10.9 и выше.

1.3.3. Windows

Согласно найденной статистике [15] на октябрь 2021 года менее 1% пользователей операционной системы Windows пользуются версиями, выпущенными раньше, чем Windows 7. К тому же версия Windows Vista не поддерживается с 2017 года, а версию Windows 7 собираются платно поддерживать до 2023 года.

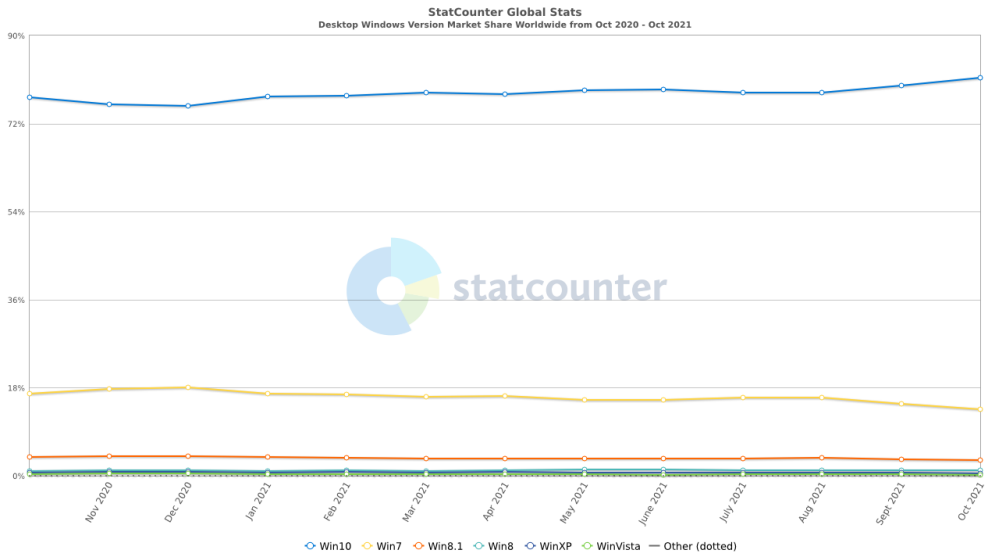

Рис. 4: Статистика использования версий Windows

В отличие от Unix-подобных операционных систем в Windows по умолчанию компиляторы не включены в операционную систему. Microsoft предоставляет официальный компилятор C++ — Visual C++. Существуют некоторые альтернативные компиляторы, такие как MinGW. Однако при использовании альтернативных компиляторов могут возникнуть проблемы с совместимостью CPython и полученной библиотеки, так как CPython скомпилирован с использованием Visual C++.

Актуальная версия Microsoft Visual C++ — 14. Microsoft Visual C++ входит в состав Microsoft Visual Studio. Microsoft Visual Studio 2015, 2017, 2019 и 2022 поддерживаются на всех достаточно популярных версиях Windows (10, 8.1, 8, 7) и используют 14 версию Microsoft Visual C++. Однако при сборке библиотеки с использованием Microsoft Visual C++ 14.20+ появится дополнительная зависимость от библиотеки vcruntime140_1.dll, в дополнение к зависимости от vcruntime140.dll. Таким образом, лучше использовать Microsoft Visual C++ 14.0–14.16 версий, чтобы избежать дополнительных зависимостей.

2. Постановка задачи

На основе проведенного обзора предметной области была поставлена цель создать pip-пакет для библиотеки mrob, совместимый с широким спектром операционных систем (GNU/Linux, Windows, macOS) и с наиболее популярными версиями Python 3.
Основными требованиями к решению являются:

- поддержка пакета на операционных системах GNU/Linux, macOS, Windows;
- поддержка pip-пакетом 3.5 – 3.10 версий Python;
- простая масштабируемость решения при появлении новых версий Python или новой функциональности библиотеки.

Для достижения цели были поставлены следующие задачи:

- провести обзор существующих решений;
- создать для библиотеки mrob пакет, который можно распространять через pip;
- настроить CI/CD для автоматической сборки и выпуска релизов пакета для воспроизводимости результата при обновлении функциональности библиотеки.

3. Ход работы

3.1. Выбор окружения сборки

При выборе окружения сборки во внимание принималась статистика использования версий операционных систем, а также доступность окружения на CI/CD платформах.

3.1.1. GNU/Linux

Было принято решение использовать докер образ manylinux [10], так как в нем базовым образом является образ дистрибутива Centos 6 (который предоставляет достаточно раннюю версию библиотеки glibc), а также в нем установлены нужные версии Python.

3.1.2. macOS

Было принято решение осуществлять сборку на macOS 10.15 и поддерживать версии начиная с OS X 10.9, так как поддержка старых версий операционных систем даёт максимальный охват потенциальных пользователей. К тому же нет необходимости осуществлять сборку на более ранних версиях macOS, чем 10.15, так как согласно статистике пользователей, использующих версии 10.9 и более ранние, всего чуть больше 1%.

3.1.3. Windows

Было принято решение осуществлять сборку на Windows Server 2019, на которой предустановен Microsoft Visual Studio Enterprise 2019, так как эта версия операционной системы подходит для обеспечения кроссплатформенности среди достаточно популярных версий Windows: 10, 8.1, 8 и 7, а также она доступна для использования в Github Actions, что полезно для автоматизации процесса сборки колеса.

3.2. Настройка сборки

Одним из популярных способов сборки является получение в качестве sdist zip-архива из исходного кода и переход от sdist к wheel путем разархивации sdist и запуска setup.py. Проблема использования в качестве сценария сборки только файла setup.py состоит в том, что pip предполагает, что любой файл setup.py напрямую зависит от setuptools — библиотеки, предназначенной для упрощения упаковки проектов Python. Хотя setuptools и имеет такой параметр как setup_requires, который указывает, что необходимо для сборки колеса и sdist, но не позволяет указать необходимость setuptools и тем более ее версии или другого аналога этого инструмента.

Стандарт PEP 518 [2] с помощью использования файла pyproject.toml позволяет указывать, какие инструменты для сборки необходимо использовать, и устанавливать их в виртуальном окружении.

Стандарт PEP 517 [14] позволяет указывать, как создать колесо и sdist из проекта, содержащего pyproject.toml.

3.3. Сборка модулей библиотеки отдельно под каждую версию Python

CMake пытается автоматически определить установленную версию Python и использовать ее. Указать конкретную версию можно с помощью переменной PYTHON_EXECUTABLE.

3.4. Решение проблемы линковки

После сборки библиотеки в бинарном файле основного модуля, mrob, указан полный путь к бинарным файлам остальных модулей. При переносе библиотеки в другую папку возникает проблема линковки — теперь пути к файлам поменялись и модуль mrob не может найти нужные файлы, от которых он зависит. Эта проблема станет актуальна, когда пользователь установит пакет на свою машину и бинарные файлы будут находиться по другому пути.

3.4.1. GNU/Linux

В операционных системах GNU/Linux решить эту проблему можно с помощью команды chrpath, меняющей rpath (— путь для поиска динамических библиотек) в бинарном файле. Полезно использование ORIGIN — специальной переменной, указывающей фактический путь к директории, в которой во время исполнения находится исполняемый файл.

Кроме того установить значение rpath равным ORIGIN можно изначально, при сборке библиотеки. Для этого необходимо установить значение переменной CMAKE_INSTALL_RPATH равным ORIGIN, а также CMAKE_BUILD_WITH_INSTALL_RPATH равным TRUE — иначе CMake проигнорирует значение предыдущей переменной.

3.4.2. macOS

В операционной системе macOS в библиотеках хранится следующая информация:

- **@executable_path** — путь до директории, откуда было вызвано приложение, использующее библиотеку;
- **@loader_path** — путь до директории, в которой находится библиотека;
- **@rpath** — путь до директории, в которой находятся динамические библиотеки, которые необходимо загрузить исходной библиотеке.

Если при сборке библиотеки дополнительно указать значение переменной CMAKE_INSTALL_RPATH равным @loader_path, то значение @rpath будет равно значению @loader_path. Это означает, что в нашем случае проблема линковки будет решена, так как во время линковки поиск динамических библиотек будет проводиться в той же директории, в которой находится исходная библиотека. Так же как и

для операционных систем GNU/Linux важно не забыть указать значение переменной CMAKE_BUILD_WITH_INSTALL_RPATH равным TRUE.

Также в macOS решить проблему линковки можно с помощью утилиты install_name_tool.

3.4.3. Windows

В Windows не возникает проблемы с линковкой основного модуля mrob с остальными модулями, так как поиск динамических библиотек автоматически осуществляется в том числе в директории, в которой находится та библиотека, динамические зависимости которой нужно найти.

Однако могут возникнуть проблемы с линковкой библиотек Visual C++. Как было упомянуто ранее, при сборке библиотеки с использованием Visual C++ 14.20+ появится дополнительная зависимость от vcruntime140_1.dll. Чтобы этой зависимости не было, нужно указать опцию компиляции -d2FH4- и опцию компановки -d2:-FH4-.

Также проблемы возникнут, если у пользователя вообще не установлен Visual C++, так как у модуля mrob есть зависимости от некоторых динамических библиотек Visual C++, например, от msvcp140.DLL — стандартной библиотеки C++, включающей в себя функции для манипуляций со строками, выделения памяти и многие другие.

Есть несколько вариантов решения данной проблемы. Во-первых, можно добавить недостающие библиотеки в папку, в которой находятся все полученные нами библиотеки. Недостатком этого метода является то, что если несколько используемых библиотек/приложений содержат одинаково называющиеся библиотеки, то результат того, какая из них будет использована, труднопредсказуем. Во-вторых, можно предложить пользователю установить специальный распространяемый пакет, в который входят библиотеки, которых может не хватать при отсутствии Visual C++. Этот вариант Microsoft называет рекомендуемым, поскольку он позволяет устанавливать самые свежие версии библиотек и не делает ничего, если они уже установлены. Также можно устанав-

ливать распространяемый пакет во время установки приложения, которому необходим Visual C++, однако в нашем случае такой возможности нет, так как pip не имеет опции установки произвольных программ перед установкой требуемых библиотек.

Проверить, что нужные библиотеки установлены, можно путем проверки с помощью команды REG QUERY значения в реестре по ключу вида HKLM\SOFTWARE[\Wow6432Node]\Microsoft\VisualStudio\14.0\VC\Runtimes\x64|x86.

3.5. Автоматическое версионирование пакета

Метаинформация о pip-пакете указывается в файле setup.cfg. В метаинформацию входит название pip-пакета, его версия, описание, лицензия, данные об авторах и многое другое. Если версию указывать в файле setup.cfg, то перед выпуском каждого нового релиза ее придется менять вручную. Чтобы не возникало ошибок с номером версии, хочется автоматизировать этот процесс.

Существует множество инструментов для автоматизации версионирования за счёт добавления тега коммиту в системе контроля версий. Был выбран инстумент setuptools-git-versioning [7], так как он обладает наибольшими преимуществами и активно поддерживается. Сводную таблицу сравнения инструментов можно изучить здесь.

3.6. Заполнение файла *__init__.py*

Файл __init__.py определяет к каким классам пользователь может получить доступ через интерфейс пакета. Заполнить его можно так:

```
from . import mrob

FGraph = mrob.FGraph
GN = mrob.GN
...
ostream_redirect = mrob.ostream_redirect
registration = mrob.registration

del mrob
```

Однако при добавлении нового модуля в библиотеку, нужно будет вручную добавлять его в __init__.py, поэтому более разумно будет использовать функции dir() и globals() при написании __init__.py файла. Функция dir(obj) возвращает все атрибуты объекта obj. Функция globals() возвращает словарь со значениями текущих глобальных переменных. В этот словарь необходимо добавить нужные атрибуты библиотеки. Итоговый __init__.py выглядит так:

```
from . import mrob

for module in dir(mrob):
  n = len(module) - 1
  if not (module[:2] == '__' and module[n:n-2:-1] == '__') and module.
  count('.') == 0:
      globals()[module] = getattr(mrob, module)

del mrob
```

3.7. Сборка колеса

Сборку колеса можно осуществить при помощи библиотеки build [9], которая позволяет выполнять сборку согласно стандарту PEP 517. Структура директории, в которой будет осуществляться сборка, должна выглядеть так:

Файлы пакета — это скомпилированный для каждой версии Python модуль mrob, а также оставшиеся скомпилированные модули библио-

теки, не зависящие от версии Python. При использовании библиотеки будет импортироваться та версия модуля mrob, которая соответствует версии интерпретатора Python. В свою очередь к модулю mrob будут линковаться остальные модули библиотеки.

3.8. Настройка CI/CD

В ходе работы было принято решение отказаться от Circle CI в пользу Github Actions, так как в отличие от Circle CI Github Actions предоставляет бесплатную возможность запуска задач на операционных системах macOS и Windows.

Для беспечения непрерывной интеграции и доставки были выделены и настроены следующие процессы.

1. Сборка библиотеки и запуск unit-тестов для проверки корректности кода, написанного на C++.
2. Сборка колёс под каждую из платформ:
   - GNU/Linux;
   - macOS;
   - Windows 32bit,
   - Windows 64bit.
3. Запуск unit-тестов с использованием полученных колёс на каждой из вышеупомянутых платформ.
4. Загрузка колёс в GitHub Releases и в PyPI.

Заключение

В ходе работы были выполнены следующие задачи.

- Проведен обзор предметной области.
- Проведен обзор существующих решений по обеспечению кросс-платформенности.
- Создан пакет для библиотеки mrob с поддержкой 3.5 – 3.10 версий Python, который можно распространять через pip для:
  - семейства операционных систем на базе ядра Linux;
  - операционной системы macOS;
  - операционной системы Windows.
- Настроен CI/CD для автоматической сборки и выпуска релизов.

Текущая версия кода доступна по данной ссылке.
Благодаря полученным результатам, виден дальнейший вектор работы со следующими поставленными задачами:

- поддержка пакета на встраиваемых устройствах;
- распространение пакета через систему управления пакетами conda.

Список литературы

[1] Apple Inc. Minimum requirements and supported SDKs. — 2021. — https://developer.apple.com/support/xcode/.

[2] Cannon Brett, Smith Nathaniel, Stufft Donald. Specifying Minimum Build System Requirements for Python Projects. — 2016. — https://www.python.org/dev/peps/pep-0518/.

[3] Circle CI. — 2011-2020. — https://circleci.com/about/.

[4] Ferrer Gonzalo, Kornilova Anastasiia, Goncharov Nikolai. MROB: Mobile Robotics library. — 2020. — https://github.com/MobileRoboticsSkoltech/mrob.

[5] Jakob Wenzel, Rhinelander Jason, Moldovan Dean. pybind11 – Seamless operability between C++11 and Python. — 2017. — https://github.com/pybind/pybind11.

[6] Loli Fabio. Linux Distributions Timeline. — 2016-2021. — https://github.com/FabioLolix/LinuxTimeline.

[7] Maxim Martynov. Automatically set package version using git tag/commit. — 2019-2021. — https://github.com/dolfinus/setuptools-git-versioning.

[8] PyPA. Packaging Python Projects. — 2013–2020. — https://packaging.python.org/tutorials/packaging-projects/.

[9] PyPA. Correct PEP 517 package builder. — 2020. — https://pypi.org/project/build/.

[10] PyPA. Manylinux — Python wheels that work on any linux (almost). — 2020. — https://github.com/pypa/manylinux.

[11] Python Software Foundation. C API Stability. — 2001-2021. — https://docs.python.org/3/c-api/stable.html.

[12] Python Software Foundation. Python Package Index. — 2020. — https://pypi.org/.

[13] Python Software Foundation. Test Python Package Index. — 2020. — https://test.pypi.org/.

[14] Smith Nathaniel J., Kluyver Thomas. A build-system independent format for source trees. — 2015. — https://www.python.org/dev/peps/pep-0517/.

[15] StatCounter. Desktop Windows Version Market Share Worldwide. — 2021. — https://gs.statcounter.com/os-version-market-share/windows/desktop/worldwide.

[16] StatCounter. Desktop macOS Version Market Share Worldwide. — 2021. — https://gs.statcounter.com/macos-version-market-share/desktop/worldwide.

[17] TIOBE. TIOBE Index for November 2021. — 2021. — https://www.tiobe.com/tiobe-index/.

[18] The Data Science Council Of America. Top 6 Programming Languages for Data Science in 2021. — 2021. — https://www.dasca.org/world-of-big-data/article/top-6-programming-languages-for-data-science-in-2021.