Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Всероссийская олимпиада школьников
«Шаг в будущее, Космонавтика»

Применение системы управления с использованием компьютерного зрения для автономной навигации аппарата

Автор: Кудряшов Игорь Сергеевич, ГБОУ 1505, г. Москва, 11класс

Научный руководитель: Метасов И. Е., аспирант кафедры СМ7 МГТУ им. Н.Э. Баумана

Москва

2018 г.

Содержание

Введение

За последние годы были созданы и получили дальнейшее развитие довольно много *opensource* инструментов в области анализа данных и машинного обучения, в том числе в сфере компьютерного зрения. Доступные на данный момент пользовательские вычислительные платформы способны обеспечить обработку видео в режиме реального времени.

Оснащение системами компьютерного зрения часто позволяет повысить точность работы аппаратов, позволяя избежать вызванных человеческим фактором ошибок, упрощает работу оператора, а также делает возможным использование аппаратов в местах, где прямое управление невозможно по причине задержки сигнала и сложности управления. Подобные системы используются во многих сферах - от промышленного производства до межпланетных станций. Внедрение автономных летательных аппаратов автоматизировать такие задачи как

- Доставка грузов
- Сканирование местности
- Патрулирование

Постановка задачи

Цель работы - разработка прототипа системы управления квадрокоптером на основе компьютерного зрения для автономной навигации, поиска или доставки объекта и самостоятельного возврата аппарата.

Для достижения поставленной цели необходимо:

- Научиться определять положение камеры в пространстве по изображению
- Провести анализ возможных алгоритмов, не требующих большой вычислительной мощности
- Разработать программное обеспечение для определения положения камеры в пространстве
- Связать полученное ПО с полетным контроллером квадрокоптера
- Проверить возможность автономного полета

Разработка алгоритмов определения положения

Способность к самостоятельной навигации – главная особенность автономного БПЛА. GPS навигация необходима для ориентации на больших расстояниях, но точность не превышает двух метров, что критично для летательных аппаратов, к тому же подобное позиционирование плохо работает в помещениях. Решением этой проблемы является компьютерное зрение, позволяющее определять положение системы в пространстве, обнаруживать сложные объекты и избегать препятствия. Позиционирование системы в пространстве – сложная и комплексная задача, ведь из двумерного изображения необходимо получить точные сведения об окружающей среде. Процесс распознавания часто основан на сравнении параметров найденного объекта с заранее подготовленными шаблонами. При распознавании с жестко закрепленной камеры часто возникают проблемы из-за особенностей среды – перекрытие объекта, перепады яркости, генерирующие шумы.

При анализе видео с дрона добавляется еще вибрация, поэтому необходимо использовать максимально надежную и простую технологию, ведь даже пары секунд достаточно, чтобы аппарат потерял ориентацию.

В качестве фиксированных позиций для определения положения я решил использовать *Aruco* метки (рис. 1). Технология представляет собой черно-белые маркеры, которые можно использовать по отдельности или же группировать в карту.

Основное преимущество этих маркеров состоит в том, что одна такая метка обеспечивает достаточное количество информации для определения положения камеры и не требует больших вычислений. Технология позволяет генерировать большие словари уникальных меток со своими id и группировать их в карты, создавая устойчивое пространство для точной навигации. Кроме того, внутренняя двоичная кодификация делает их особенно надежными, что позволяет применять методы обнаружения ошибок и коррекции.

Я рассматривал другие возможные методы - в том числе *SVO* (*Fast Semi-Direct Monocular Visual Odometry*), показывающий очень высокую точность, но данная технология оказалась гораздо сложнее в настройке и требует большей вычислительной мощности. Использование в качестве маркеров светодиодов усложняет конструкцию и уменьшает надежность.

Все вычисления проводятся на одноплатном ПК *Raspberry* PI 3B. Данная платформа была выбрана из-за простоты использования и большого количества *opensource* инструментов.

Генерация маркеров

Маркер представляет собой матрицу, заполненную черными и белыми пикселями. Границы маркера всегда черные, а размер ячеек в каждой группе маркеров фиксирован.

Алгоритм основывается на стохастическом процессе генерации маркеров и добавлении их в словарь (рис. 2).

Подробнее с алгоритмом можно ознакомиться в приложенной статье (Список Литературы, пункт 6).

Распознавание

На рисунке 3 показано поэтапная обработка изображения

(B) После получения изображения (A) происходит переход в градации серого – все пиксели делятся либо на черные, либо на белые.

(C) Далее происходит поиск контуров – в качестве детектора используется алгоритм Сатоши-Сузуки (Список литературы п.7) топологического анализа границ, т.к. он гораздо быстрее детектора Кенни.

(D) Лишние контуры отсекаются, происходит фильтрация.

(E,F) После фильтрации происходит детальный анализ маркера. Каждый маркер делится сеткой и ячейки заполняется 0 и 1 в зависимости от цвета пикселя. После первой итерации рассматриваются объекты, границы которых заполнены 0 (черный цвет). После этого анализируется внутренняя матрица. Матрица метки проверяется на все 4 возможных поворота (90 градусов). Для ускорения анализа словарь меток сортируется с помощью сбалансированного бинарного дерева, что позволяет добиться логарифмической сложности.

После того, как маркер обнаружен, можно определить положение камеры относительно меток - для этого достаточно посмотреть на изначальную позицию маркера в словаре. Нахождение координат происходит с помощью использования модели «камеры обскуры» (рис.4).

В этой модели представление сцены формируется путем проецирования трехмерных точек в плоскость изображения с использованием преобразования перспективы.

$s\,m' = A[R|t]M'$ или

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$, где

- (X, Y, Z) являются координатами трехмерной точки в координатном пространстве

- (u, v) являются координатами точки проекции в пикселях
- A является матрицей камеры
- (cx, cy) является основной точкой, которая обычно находится в центре изображения
- fx, fy - фокусные расстояния, выраженные в пикселях.

Код ПО был написан на *python* с использованием библиотеки *OpenCV* и сервисов *ROS*.

На рисунке 5 показан пример программы зависания над меткой. Сначала задаются параметры ROS сервисов – сервер для отладки, начальные параметры полета.

Функция *navigate* отвечает за положение коптера – вычисляет расстояние и отдает команды контроллеру *pixhawk PX4* через *mavlink*. На вход принимаются координаты, система координат и скорость (рис 5.2).

arm = состояние моторов (есть / нет питание)

Frame_id - система координат, относительно которой дрон считает свое положение

- *fcu_horiz* - отсчет идет относительно начального положения дрона (место старта)
- *aruco_map* – коптер ориентируется по загруженной карте aruco маркеров

Так как алгоритм не требует затратных вычислений, дрон способен обрабатывать изображения даже с двух камер одновременно с частотой >= 20 кадров в секунду.

Посадка происходит при постепенном снижении высоты после стабилизации аппарата над меткой. При потере метки из вида аппарат будет удерживать свое текущее положение, а в процессе посадки потеря метки практически не отразится.

Тестирование ПО для управления БПЛА

Полетным контроллером дрона выступает *Pixhawk PX4*. Управление моторами осуществляется PID регулятором с экспериментально подобранными коэффициентами.

Для взаимодействия Raspberry и Pixhawk используется протокол mavlink, который преобразует команды с Raspberry Pi в низкоуровневые пакеты, аналогичные отданным с пульта РУ. Управление Pixhawk осуществляется через MAVROS[2].

MAVLink - низкоуровневый протокол, который используется для обмена информацией с беспилотным летательным аппаратом. Пакет в этом протоколе имеет длину от 8 до 263 байт. В проекте для упрощения обмена данных используется высокоуровневая утилита MAVROS, являющимся модулем (Node) ROS (рис 6 - распределение управления в аппарате).

Были проведены тесты автономного зависания и передвижения, используя карту из aruco меток.

На рисунке 7 показан вид с камеры аппарата при полете над полем маркеров(рис. 8 - сам аппарат).

Заключение

Результаты работы - проведен анализ наиболее оптимальных алгоритмов и способов определения положения в пространстве. Выбранные методы были объединены в полученном ПО для автономной навигации. Аппарат способен совершать автономный взлет и посадку, навигацию по заданной карте с точностью до сантиметров и облет небольшой территории.

Планируется добавить GPS навигацию, оснастить аппарат дополнительными сенсорами для надежности и внедрить их.

Список литературы

1. Документации библиотеки OpenCV, версия 2.4.13 [Электронный ресурс] // http://docs.opencv.org/2.4.13 (Дата обращения 05.12.2018)
2. Документация MAVROS offboard для контроллера PX4 https://dev.px4.io/en/ros/mavros_offboard.html
3. Документация технологии Aruco https://docs.opencv.org/3.2.0/d9/d6a/group__aruco.html
4. Документация протокола mavlink для контроллера PX4 https://dev.px4.io/en/middleware/mavlink.html
5. Документация сервиса ROS [Электронный ресурс] // http://wiki.ros.org/mavros
6. S.Garrido-JuradoR.Muñoz-SalinasF.J.Madrid-CuevasM.J.Marín-Jiménez Automatic generation and detection of highly reliable fiducial markers under occlusion (Elsevier, 2014)
7. S. Suzuki, K. Abe, Topological structural analysis of digitized binary images by border following, Comput. Vis. Graph. Image Process. 30 (1) (1985) 32–46.

Приложения

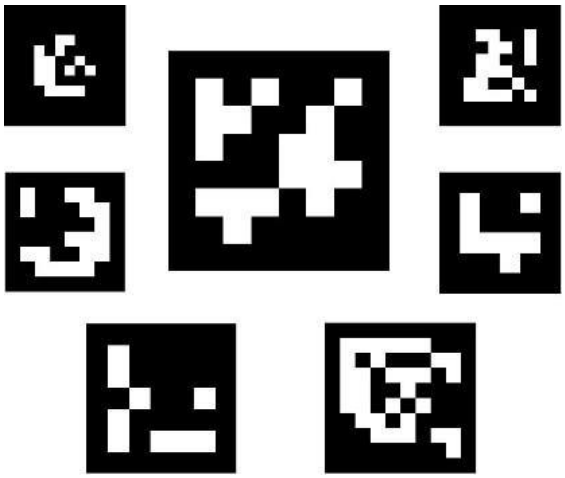

Рис. 1. Пример Aruco метки

$\mathfrak{D} \leftarrow \varnothing$ # *Reset dictionary*
$\tau \leftarrow \tau^0$ # *Initialize target distance, see Section 3.4*
$\varrho \leftarrow 0$ # *Reset unproductive iteration counter*
while $\mathfrak{D}$ has not desired size **do**
  Generate a new marker m # *Section 3.2*
  if distance of m to elements in $\mathfrak{D}$ is $\geq \tau$ **then**
    $\mathfrak{D} \leftarrow \mathfrak{D} \cup m$ # *Add to dictionary*
    $\varrho \leftarrow 0$
  else
    $\varrho \leftarrow \varrho + 1$ # *It was unproductive*
    # *maximum unproductive iteration reached?*
    if $\varrho = \psi$ **then**
      $\tau \leftarrow \tau - 1$ # *Decrease target distance*
      $\varrho \leftarrow 0$
    end if
  end if
end while

Рис. 2. Алгоритм генерации маркеров и заполнения словаря

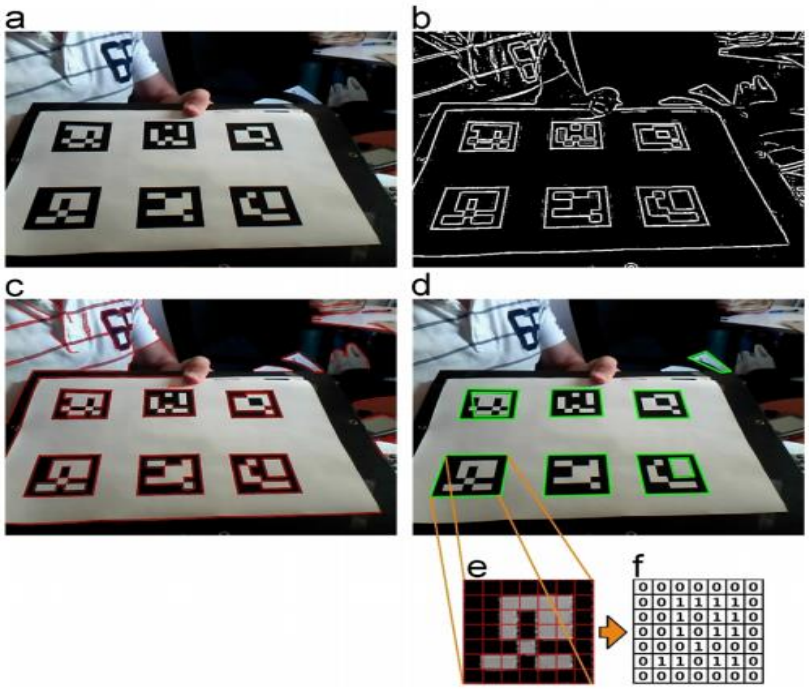

Рис. 3. Поэтапная обработка изображения

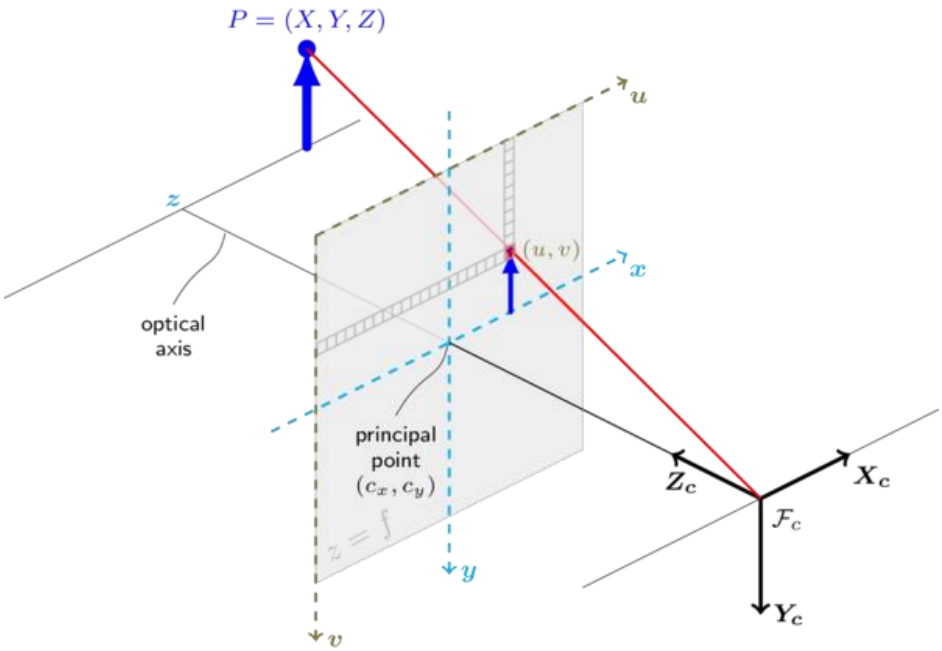

Рис. 4. Модель «камеры обскуры»

```python
import rospy
from clever import srv
from std_srvs.srv import Trigger
import time
from mavros_msgs.srv import SetMode

rospy.init_node('test_up_down')

navigate = rospy.ServiceProxy('/navigate', srv.Navigate)

set_position = rospy.ServiceProxy('/set_position', srv.SetPosition)
set_position_yaw_rate = rospy.ServiceProxy('/set_position/yaw_rate', srv.SetPositionYawRate)

set_velocity = rospy.ServiceProxy('/set_velocity', srv.SetVelocity)
set_velocity_yaw_rate = rospy.ServiceProxy('/set_Velocity/yaw_rate', srv.SetVelocityYawRate)

set_attitude = rospy.ServiceProxy('/set_attitude', srv.SetAttitude)
set_attitude_yaw_rate = rospy.ServiceProxy('/set_attitude/yaw_rate', srv.SetAttitudeYawRate)

set_rates_yaw = rospy.ServiceProxy('/set_rates/yaw', srv.SetRatesYaw)
set_rates = rospy.ServiceProxy('/set_rates', srv.SetRates)

set_mode = rospy.ServiceProxy('/mavros/set_mode', SetMode)

release = rospy.ServiceProxy('/release', Trigger)
#настройка параметров ROS модулей
navigate(x=0, y=0, z=2, speed=1, frame_id='fcu_horiz', auto_arm=True)  # взлет

time.sleep(5) # 5 секундное ожидание

navigate(x=0, y=0, z=-1.6, speed=0.5, frame_id='fcu_horiz')

time.sleep(5) # 5 секундное ожидание
```

Рис. 5.1. Код программы зависания над меткой

```python
def get_navigate_setpoint(stamp, start, finish, start_stamp, speed):
    distance = math.sqrt((finish.z - start.z)**2 + (finish.x - start.x)**2 + (finish.y - start.y)**2)
    time = rospy.Duration(distance / speed)
    k = (stamp - start_stamp) / time
    time_left = start_stamp + time - stamp

    if BRAKE_TIME and time_left < BRAKE_TIME:
        # time to brake
        time_before_braking = time - BRAKE_TIME
        brake_time_passed = (stamp - start_stamp - time_before_braking)

        if brake_time_passed > 2 * BRAKE_TIME:
            # finish
            k = 1
        else:
            # brake!
            k_before_braking = time_before_braking / time
            k_after_braking = (speed * brake_time_passed.to_sec() - brake_time_passed.to_sec() ** 2 * speed / 4 / BRAKE_TIME.to_sec()) / di
            k = k_before_braking + k_after_braking

    k = min(k, 1)

    p = Point()
    p.x = start.x + (finish.x - start.x) * k
    p.y = start.y + (finish.y - start.y) * k
    p.z = start.z + (finish.z - start.z) * k
    return p
```

Рис 5.2 – функция, отвечающая за навигацию аппарата

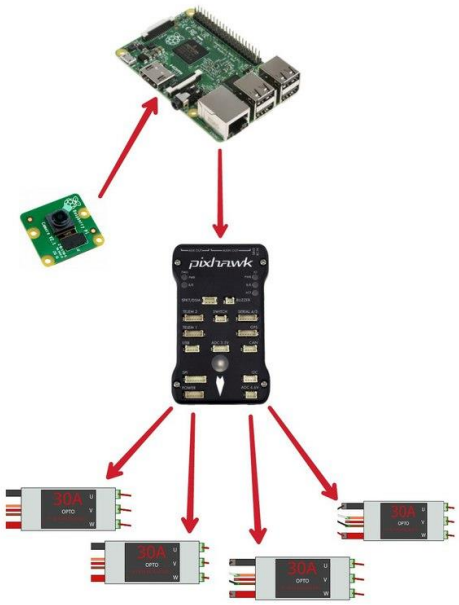

рис.6.

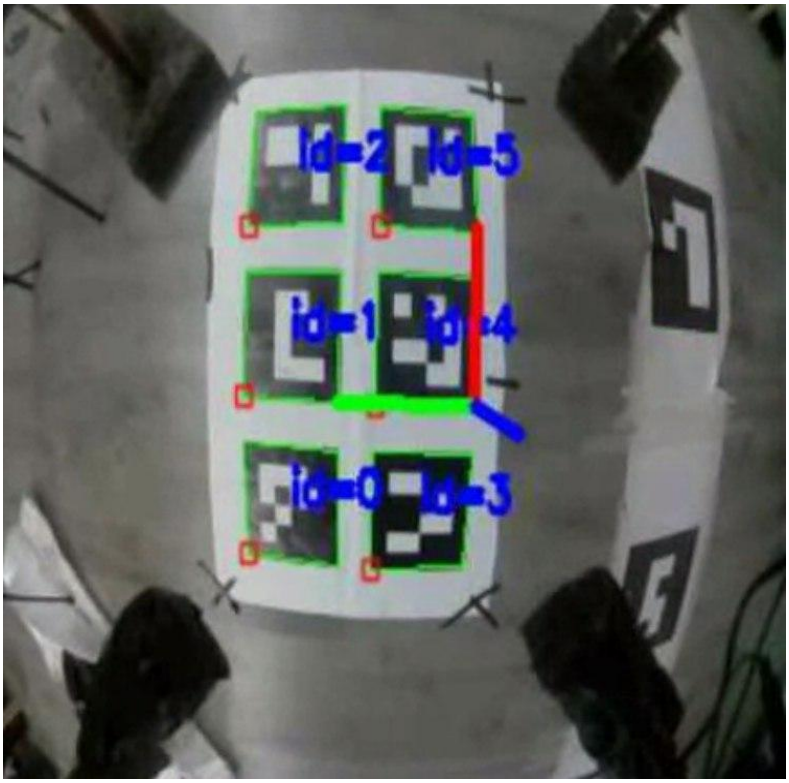

Рис.7. Вид с камеры аппарата

Рис.8. Используемый квадрокоптер