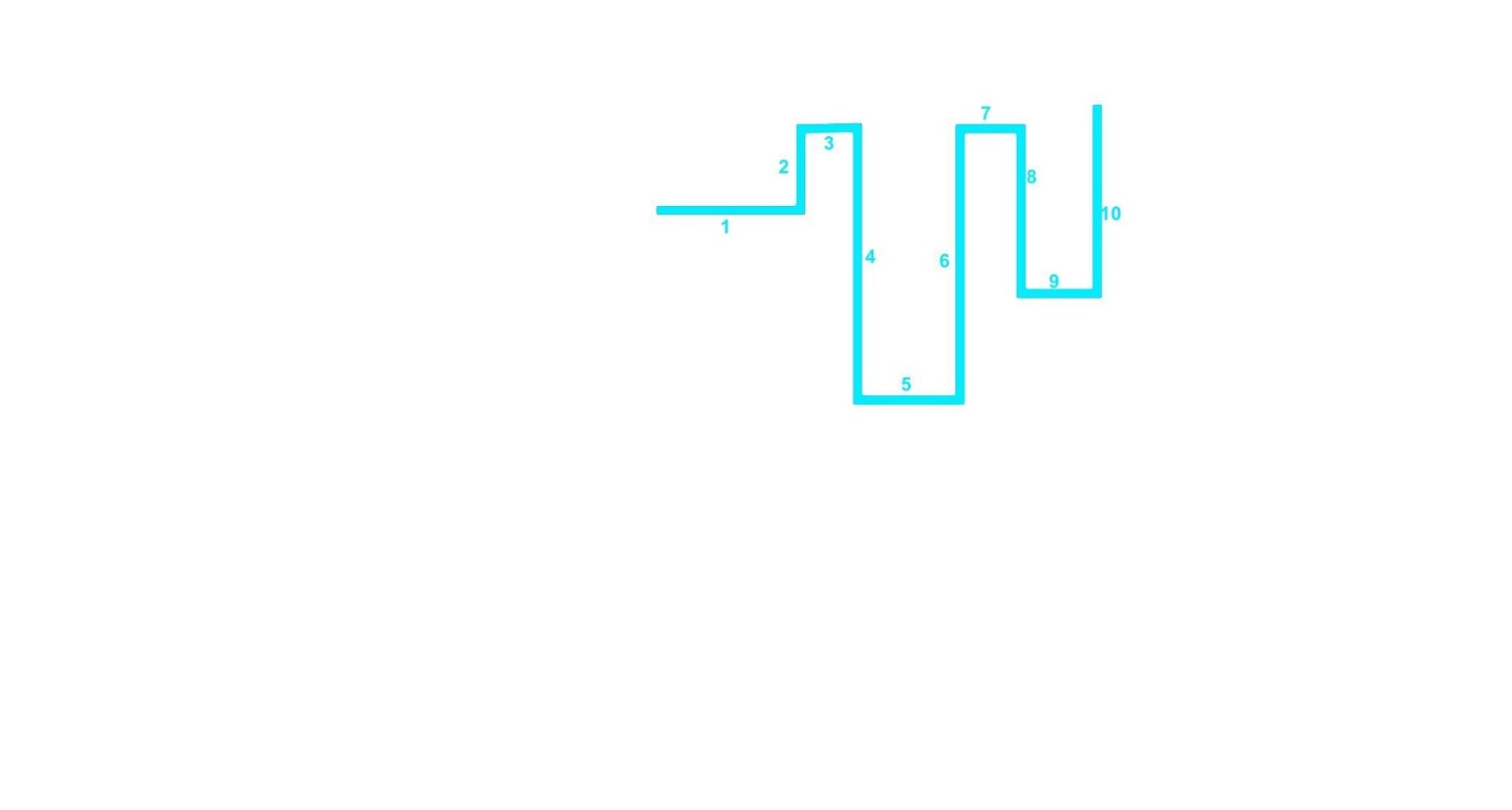
Крыжовник

На протяжении всего финала НТО 2023 наша команда долго и упорно трудилась над написанием программного кода для автономного полёта коптера, обнаружения очагов возгорания и пострадавших.

Краткое описание решаемой задачи:

1. **Взлет:**
   1. Совершить автономный взлет с зоны «Н».
   2. Установить значение высоты дрона не более 1.5 метров, возможно изменение в течение миссии, но только в меньшую сторону (не поднимать дрон выше стены).
2. **Автоматизированный поиск очагов возгорания в помещении при помощи автономного квадрокоптера:**
   1. Прилететь к точке начала мониторинга (постоянна, известна).
   2. Начать движение по любой траектории, позволяющей обнаружить возгорания, находящиеся внутри помещения (светло-серая зона на Рис. 1).
      1. В случае выхода проекции квадрокоптера за пределы помещения на 5 секунд и более, считается, что дрон окончил мониторинг и дальнейших действий не планируется. Дальнейшие баллы не начисляются.
   3. Определить длину каждой стены, согласно нумерации ниже: 2.3.1 Отобразить каждую стену, привязанную к системе координат, в системе визуализации любым возможным методом (marker array, point cloud, и т.д.; важно чтобы была возможность различить стены на экране)

2.4Обнаружить все возгорания (при их наличии).

* + 1. Отобразить возгорания, привязанные к системе координат в системе визуализации (Rviz или аналоге, используя marker array), форма и размер значение не имеет, цвет - оранжевый.
    2. Сделать запрос на сервер с координатами возгорания. Формат запроса указан в [OpenAPI документации](http://65.108.222.51/docs).
    3. В зависимости от полученного результата определяем класс пожара (данная функция необходима для выполнения дальнейшего задания).
    4. Вывести в терминал сообщение о координатах обнаруженных возгораний относительно aruco\_map типе пожара и хранящемся в нем веществе. Формат: *fire: <x> <y> <material> <class>В пункте <material> указывается конкретный тип.*
    5. 2.4.5 После обнаружения возгорания и определения его типа, необходимо выполнить сброс необходимой капсулы. Класс А - цилиндр, класс В - куб.
  1. Обнаружить пострадавших в пожаре рабочих. На всех рабочих были надеты каски синего цвета.
     1. Вывести в терминал координаты обнаруженных пострадавших Формат: *injured: <x> <y>*
     2. Отобразить местонахождение пострадавших с привязкой к системе координат в системе визуализации (Rviz или аналоге, используя marker array), форма и размер значение не имеет, цвет -синий.
     3. Определить ближайший очага возгорания.
  2. Продолжить мониторинг, в случае обнаружения иных возгораний повторить пункт 2.3-2.5;

1. **По окончанию мониторинга помещения, вернуться в зону «Н»;**
2. **Совершить автономную посадку в пределах зоны «H».**

Наш код решения данной задачи на языке программирования Python с пояснениями:

|  |
| --- |
| import rospy |
|  | import math |
|  | import cv2 as cv |
|  | import numpy as np |
|  | from cv\_bridge import CvBridge |
|  | from sensor\_msgs.msg import Image |
|  | from clover.srv import SetLEDEffect |
|  | from clover import srv |
|  | from std\_srvs.srv import Trigger |
|  | #импорт библиотек |
|  |  |
|  |  |
|  | rospy.init\_node('computer\_vision\_sample') |
|  | bridge = CvBridge() |
|  |  |
|  | get\_telemetry = rospy.ServiceProxy('get\_telemetry', srv.GetTelemetry) |
|  | navigate = rospy.ServiceProxy('navigate', srv.Navigate) |
|  | navigate\_global = rospy.ServiceProxy('navigate\_global', srv.NavigateGlobal) |
|  | set\_position = rospy.ServiceProxy('set\_position', srv.SetPosition) |
|  | set\_velocity = rospy.ServiceProxy('set\_velocity', srv.SetVelocity) |
|  | set\_attitude = rospy.ServiceProxy('set\_attitude', srv.SetAttitude) |
|  | set\_rates = rospy.ServiceProxy('set\_rates', srv.SetRates) |
|  | land = rospy.ServiceProxy('land', Trigger) |
|  |  |
|  | set\_effect = rospy.ServiceProxy('led/set\_effect', SetLEDEffect) |
|  |  |
|  | hsv\_min = np.array((100, 100, 100), np.uint8) |
|  | hsv\_max = np.array((180, 255, 200), np.uint8) |
|  |  |
|  | def image\_callback2(data): #функция для выделения контуров и цетра фигуры |
|  | global cv\_image |
|  | cv\_image = bridge.imgmsg\_to\_cv2(data, 'bgr8') |
|  |  |
|  | hsv = cv.cvtColor(cv\_image, cv.COLOR\_BGR2HSV) |
|  | thresh = cv.inRange(hsv, hsv\_min, hsv\_max ) |
|  | contours, hierarchy = cv.findContours(thresh.copy(), cv.RETR\_TREE, cv.CHAIN\_APPROX\_NONE) |
|  | contours = list(filter(lambda x: cv.contourArea(x) > 1, contours)) |
|  | black\_white = cv.bitwise\_and(cv\_image, cv\_image, mask=thresh) |
|  | if len(contours) != 0: |
|  | cont = max(contours, key=cv.contourArea) |
|  |  |
|  | epsilon = 0.1\*cv.arcLength(cont,True) |
|  | approx = cv.approxPolyDP(cont,epsilon,True) |
|  |  |
|  | cv.drawContours(cv\_image, cont, -1, (255,0,0), 3, cv.LINE\_AA) |
|  | moments = cv.moments(cont, 1) |
|  | dM01 = moments['m01'] |
|  | dM10 = moments['m10'] |
|  | dArea = moments['m00'] |
|  | if dArea > 100: |
|  | x = int(dM10 / dArea) |
|  | y = int(dM01 / dArea) |
|  | cv.circle(cv\_image, (x, y), 10, (0,255,254), -1) |
|  | cv.putText(cv\_image, (str(x)), (x, y), cv.FONT\_HERSHEY\_SIMPLEX, 1, (0, 100, 200), 2) |
|  | cv.putText(cv\_image, (str(y)), (x, y-30), cv.FONT\_HERSHEY\_SIMPLEX, 1, (0, 100, 200), 2) |
|  |  |
|  | image\_pub.publish(bridge.cv2\_to\_imgmsg(cv\_image, 'bgr8')) |
|  |  |
|  |  |
|  | def image\_callback(data): #функция для унавания распознаваемых он с помощью hsv |
|  | global cv\_image |
|  | cv\_image = bridge.imgmsg\_to\_cv2(data, 'bgr8') |
|  | hsv = cv.cvtColor(cv\_image, cv.COLOR\_BGR2HSV) |
|  | thresh = cv.inRange(hsv, hsv\_min, hsv\_max ) |
|  | contours, hierarchy = cv.findContours(thresh.copy(), cv.RETR\_TREE, cv.CHAIN\_APPROX\_SIMPLE) |
|  | contours = list(filter(lambda x: cv.contourArea(x) > 1, contours)) |
|  | black\_white = cv.bitwise\_and(cv\_image, cv\_image, mask=thresh) |
|  | if len(contours) != 0: |
|  | cont = max(contours, key=cv.contourArea) |
|  | cv.drawContours(cv\_image, cont, -1, (255,0,0), 3, cv.LINE\_AA) |
|  | moments = cv.moments(cont, 1) |
|  | dM01 = moments['m01'] |
|  | dM10 = moments['m10'] |
|  | dArea = moments['m00'] |
|  | if dArea > 100: |
|  | x = int(dM10 / dArea) |
|  | y = int(dM01 / dArea) |
|  | cv.circle(cv\_image, (x, y), 10, (0,255,254), -1) |
|  | cv.putText(cv\_image, (str(x)), (x, y), cv.FONT\_HERSHEY\_SIMPLEX, 1, (0, 100, 200), 2) |
|  | cv.putText(cv\_image, (str(y)), (x, y-30), cv.FONT\_HERSHEY\_SIMPLEX, 1, (0, 100, 200), 2) |
|  |  |
|  | image\_pub.publish(bridge.cv2\_to\_imgmsg(black\_white, 'bgr8')) |
|  |  |
|  |  |
|  | def navigate\_wait(x=0, y=0, z=1.5, yaw=0, speed=1, frame\_id='aruco\_map', auto\_arm=False, tolerance=0.2): |
|  | navigate(x=x, y=y, z=z, yaw=yaw, speed=speed, frame\_id=frame\_id, auto\_arm=auto\_arm) |
|  |  |
|  | while not rospy.is\_shutdown(): |
|  | telem = get\_telemetry(frame\_id='navigate\_target') |
|  | if math.sqrt(telem.x2 + telem.y2 + telem.z\*\*2) < tolerance: |
|  | break |
|  | rospy.sleep(0.2) |
|  |  |
|  |  |
|  | def square(side=2): (полёт по квадрату) |
|  | navigate\_wait(side, 0, 0, frame\_id='navigate\_target') |
|  | telemetry = get\_telemetry() |
|  | set\_effect(r=255, g=0, b=0) |
|  | print(telemetry.x, telemetry.y, telemetry.z) |
|  | navigate\_wait(0, side, 0, frame\_id='navigate\_target') |
|  | set\_effect(r=0, g=255, b=0) |
|  | telemetry = get\_telemetry() |
|  | print(telemetry.x, telemetry.y, telemetry.z) |
|  | navigate\_wait(-side, 0, 0, frame\_id='navigate\_target') |
|  | set\_effect(r=0, g=0, b=255) |
|  | telemetry = get\_telemetry() |
|  | print(telemetry.x, telemetry.y, telemetry.z) |
|  | navigate\_wait(0, -side, 0, frame\_id='navigate\_target') |
|  | set\_effect(r=100, g=100, b=0) |
|  |  |
|  |  |
|  | rospy.Subscriber('main\_camera/image\_raw', Image, image\_callback2, queue\_size=1) |
|  | image\_pub = rospy.Publisher('~debug', Image, queue\_size = 1) |
|  | rospy.spin() |