**Тестовое задание**

**на позицию Python developer**

# **ЗАДАЧА 1**

### **Формулировка:**

Написать функцию, осуществляющую управление движением мобильного объекта, движущегося в двумерной плоскости. Функция принимает на вход текущее состояние такого объекта и координаты цели движения, на выход предоставляет линейную и угловую скорости, позволяющие приблизится к требуемой цели. Функция вызывается с некоторым (малым) периодом и предоставляет значения скоростей, необходимых применить к управляемому объекту в вызванный момент времени.

### **Прототип функции:**

def control(state:List[float], target:Tuple[float,float],

max\_vel:float, max\_ang:float)

-> Tuple[float,float]

...

return (v,w)

### **Описание входных данных:**

state:List[float] - массив состояния мобильного объекта управления, состоящий из 4 элементов:

state[0]- x координата объекта управления;

state[1] - y координата объекта управления;

state[2] - угол yaw ориентации на плоскости;

state[3]- текущая скорость объекта управления;

target:Tuple[float,float]- координаты (x,y) цели движения;

max\_vel:float - ограничение поступательной скорости, быстрее которой разгоняться нельзя;

max\_ang:float - ограничение скорости вращения, быстрее которой совершать обороты вокруг себя нельзя.

### **Описание выходных данных:**

Tuple[float,float]- значения поступательной скорости V и скорости вращения вокруг своей оси W для достижения координат цели.

### **Примерный формат использования функции:**

for target in flight\_task:

while dist(state, target) > 0.1:  
v,w = control(state, target, 1, 1)  
state[0] += math.cos(state[2])\*v\*delta\_t  
state[1] += math.sin(state[2])\*v\*delta\_t  
state[2] += w\*delta\_t  
state[3] = v

sleep(delta\_t)

### **Допущения:**

* Объекту управления нет необходимости избегать препятствия, всё пространство свободно.
* Подлетать к цели полётного задания можно с произвольного направления.
* Объект управления выполняет команды с невообразимой точностью, пролетая за delta\_t ровно столько, сколько предсказывает теория.

# **ЗАДАЧА 2**

### **Формулировка:**

Создать систему связанных по сетевому протоколу (tcp и/или udp) клиента и сервера. Один из которых считает изображение с какого-либо расположения в операционной системе и отправляет его в socket, откуда оно прочитаутся вторым участником и выводится на экран.

### **Требования:**

* Клиент и сервер должны обладать GUI, созданном с помощью библиотеки tkinter.
* Клиент на форме должен иметь диалоговое окно выбора файла-изображения для отправки, а также текстовое поле, указывающее порт сокета связи.
* Клиент и Сервер должны иметь функционал (кнопки или другие поля), позволяющие настроить связь между ними.
* Сервер на форме должен отрисовывать приходящие изображения с клиента.

### **Допущения:**

* Клиент не останавливает свою работу после передачи одного изображения, но сохраняет интерфейс для загрузки и передачи следующего изображения.
* Клиент может отправлять только изображения, никакие другие файлы для передачи не предполагаются.