

UAV-VLA: Vision-Language-Action System for Large Scale Aerial Mission Generation

Oleg Sautenkov Skoltech Moscow, Russia oleg.sautenkov@skoltech.ru	Yasheerah Yaqoot Skoltech Moscow, Russia yasheerah.yaqoot@skoltech.ru	Artem Lykov Skoltech Moscow, Russia artem.lykov@skoltech.ru	Muhammad Ahsan Mustafa Skoltech Moscow, Russia ahsan.mustafa@skoltech.ru
--	--	--	---

Grik Tadevosyan Skoltech Moscow, Russia grik.tadevosyan@skoltech.ru	Aibek Akhmetkazy Skoltech Moscow, Russia aibek.akhmetkazy@skoltech.ru	Miguel Altamirano Cabrera Skoltech Moscow, Russia m.altamirano@skoltech.ru	Mikhail Martynov Skoltech Moscow, Russia mikhail.martynov@skoltech.ru
--	--	---	--

Sausar Karaf
Skoltech
Moscow, Russia
sausar.karaf@skoltech.ru

Dzmitry Tsetserukou
Skoltech
Moscow, Russia
d.tsetserukou@skoltech.ru

主讲人：郭周鹏
日期：2025.9.26



Introduction-背景与挑战

研究背景:

- 无人机 (UAV) 在 农业巡检、城市管理、灾害救援、监控 等领域得到广泛应用;
- 随着任务复杂度增加, 人机交互需求更高;
- 传统控制方式依赖人工遥控或复杂的任务编程, 对非专业用户门槛高;
- 多模态大模型 (LLM/VLM) 的发展, 使得 “自然语言 → 行动指令” 的新范式成为可能。

核心挑战:

- 自然语言与无人机控制的鸿沟: 用户的指令是模糊的自然语言, 而无人机执行需要严格的航点与动作命令。
- 跨模态融合难: 需要把语言描述和卫星图像等视觉数据对齐, 并准确找到目标。
- **泛化能力不足:** 现有许多 VLA/VLN 模型依赖大规模配对数据集, 难以在开放环境中零样本应用。
- 任务生成效率低: 人工任务规划耗时长, 不适合应急或大规模任务。



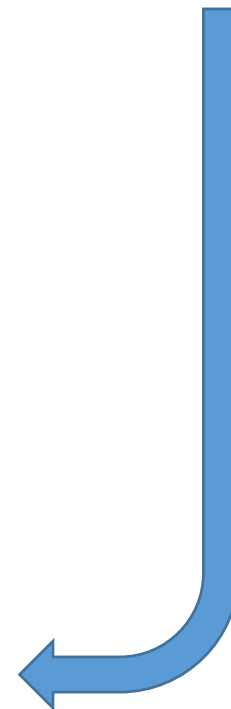
Introduction-挑战与研究目标

现有方法的局限性:

- **手动方式:** 依赖人工绘制航点, 费时费力 (论文实验中人工操作 30 个任务需要 35 分钟)。
- **传统导航方法:** 多集中于“路径规划”, 但无法表达复杂动作 (如“绕圈” “拍照” “返航”)。
- **已有 VLA/VLN 模型:**
 1. 输出多为路径点, 而非无人机可直接执行的动作文件。
 2. 多数基于室内/小规模环境, 缺乏**大尺度卫星图场景**的任务生成能力;
 3. 严重依赖标注数据, **难以迁移**到新任务或新场景;

理想目标:

- 模型能够 **零样本、泛化** 到**全球范围**的**卫星**影像任务;
- 输出应能 直接下发给无人机执行 (无需人工二次转换) -**VLA**。
- 生成任务既要覆盖目标识别、路径生成、动作定义, 又要高效、接近人工水平;
- 实现用户能够通过自然语言直接描述任务, 系统自动生成完整的无人机飞行任务;





Introduction-贡献与优势

【模型维度】：

- **提出 UAV-VLA 系统：集成 Vision-Language Model (VLM) + GPT 的任务生成框架。**
- **核心模块：**
 1. 目标提取 (Goal Extracting GPT)：解析用户自然语言指令，提取任务目标。
 2. 目标搜索 (Object Search VLM)：在卫星图像中定位任务目标。
 3. 动作生成 (Action Generation GPT)：将目标点转换为地理坐标，生成 UAV 可执行的动作序列 (MAVProxy 任务文件)。

贡献凝练：

- ① 实现在大尺度**卫星图像**场景下 **“通过自然语言直接描述任务，系统自动生成完整的无人机飞行任务”**。 ->**卫星场景 -和-VLA**
- ② UAV-VLA不需要任何样本->**零样本挑战->没有对模型做微调，直接拿来用**

验证与优势【验证维度】：

- 在**新构建的 UAV-VLPA-nano-30 基准数据集**（涵盖城市、乡村、交通、水体等多类型大尺度卫星图场景->**多样场景->泛化挑战**）上验证；
- 实验表明系统生成任务 平均耗时 5 分 24 秒，比人工快 6.5 倍；
- 生成路径的平均空间误差为 34.22 m，路径长度仅比人工长 21.6%；

2

Method-步骤1-GPT

该论文方法总结如下：

1. 输入指令：用户用自然语言描述飞行任务。
2. 提取目标：GPT理解指令并抓取关键目标。
- 3.1. 定位目标：视觉模型在卫星图像中找到目标位置。
- 3.2. 坐标转换：把图像坐标映射成真实经纬度。
4. 生成动作：GPT规划无人机具体飞行动作序列。
5. 输出任务：生成飞行计划文件，无人机可直接执行。

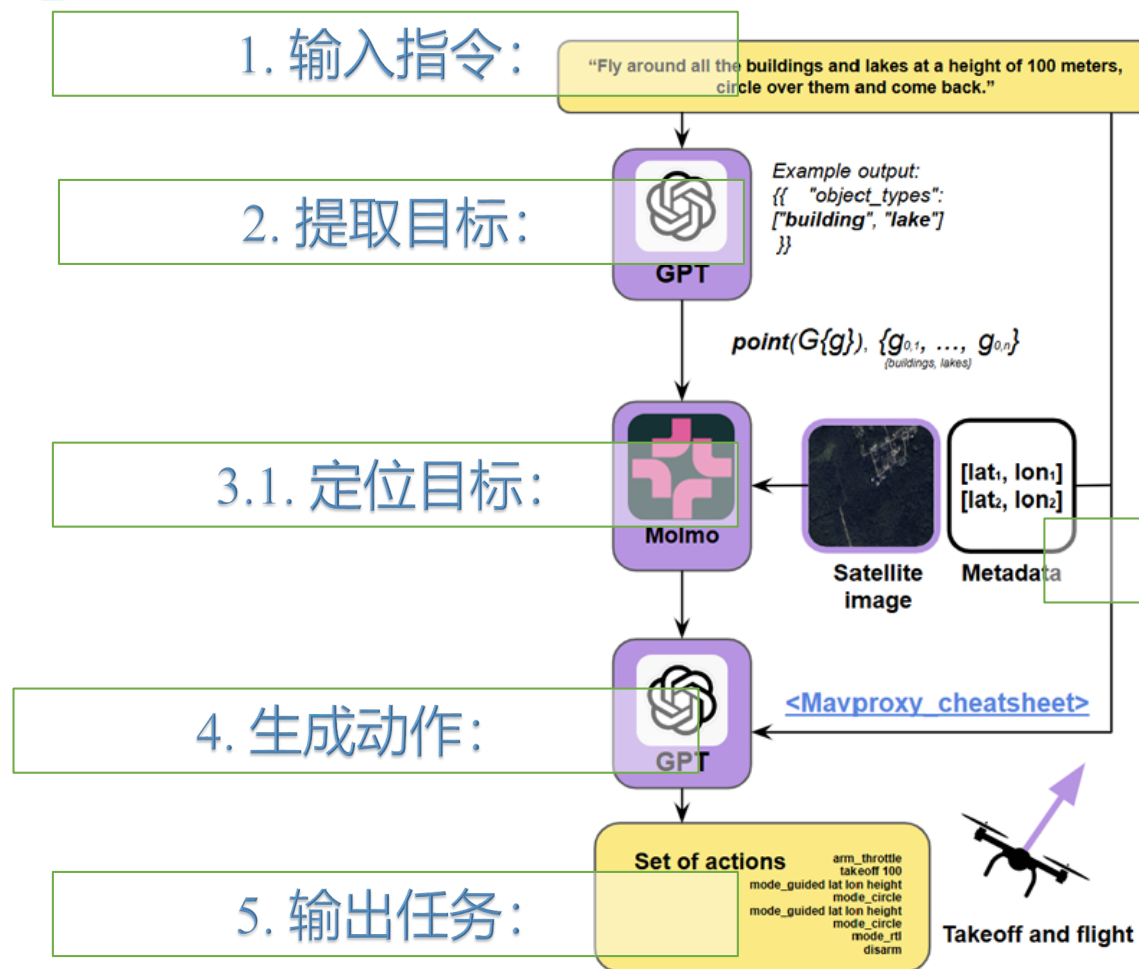


Fig. 1: The pipeline of the UAV-VLA system.

2

Method-步骤1-GPT

输入指令，并确定我们要到达的关键目标

实例：
输入：

```
{ "raw_text": "从起点 A 起飞，飞到 B 点，在 B 点上空沿半径 20 m 绕一圈（高度 100 m，拍照），然后飞往 C 点并返航。", "meta": { "user_id": "u123", "time": "2025-09-23T19:00:00+08:00" } }
```

输出：->结构化目标->对接Molmo->符合Molmo输入的数据格式

```
{ "goals": [
  { "id": "A", "type": "start", "note": "home position (if known)" },
  { "id": "B", "type": "target", "action": "circle", "params": { "radius_m": 20, "alt_m": 100, "take_photo": true } },
  { "id": "C", "type": "target", "action": "go_to", "params": { "alt_m": 100 } },
  { "id": "finish", "type": "return_to_launch" }
], "constraints": { "max_flight_time_min": 30, "no_fly_zones": [] } }
```

The **goal extracting GPT module** parses I into a set of goals:

$$G = GPT(I) = \{g_1, g_2, \dots, g_n\}, \quad (2)$$

where G contains goals derived from the instruction, tailored to the task.

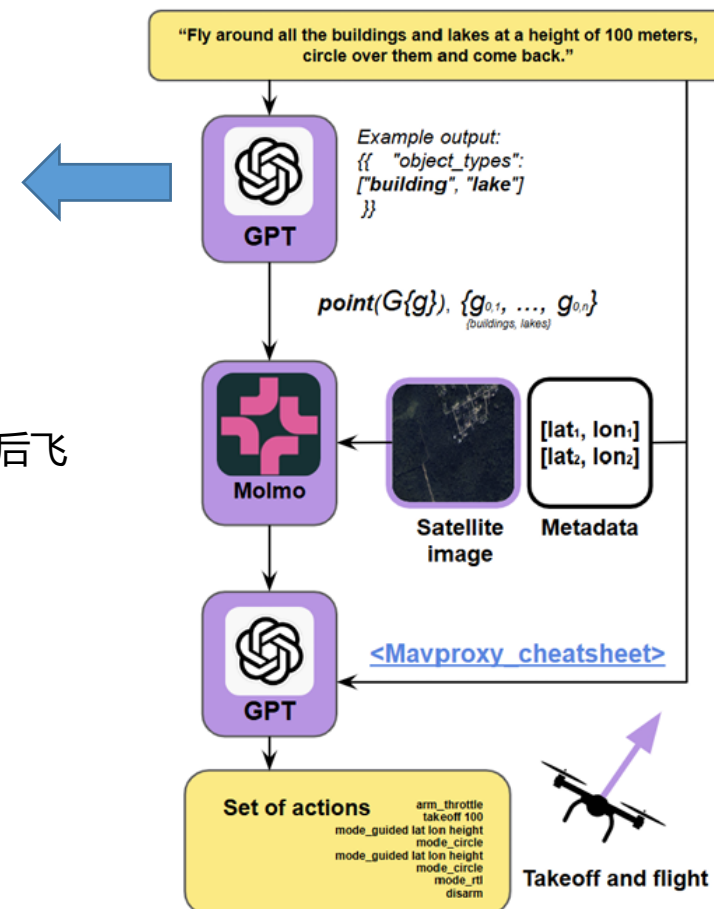


Fig. 1: The pipeline of the UAV-VLA system.

2

Method-步骤2.1-Molmo

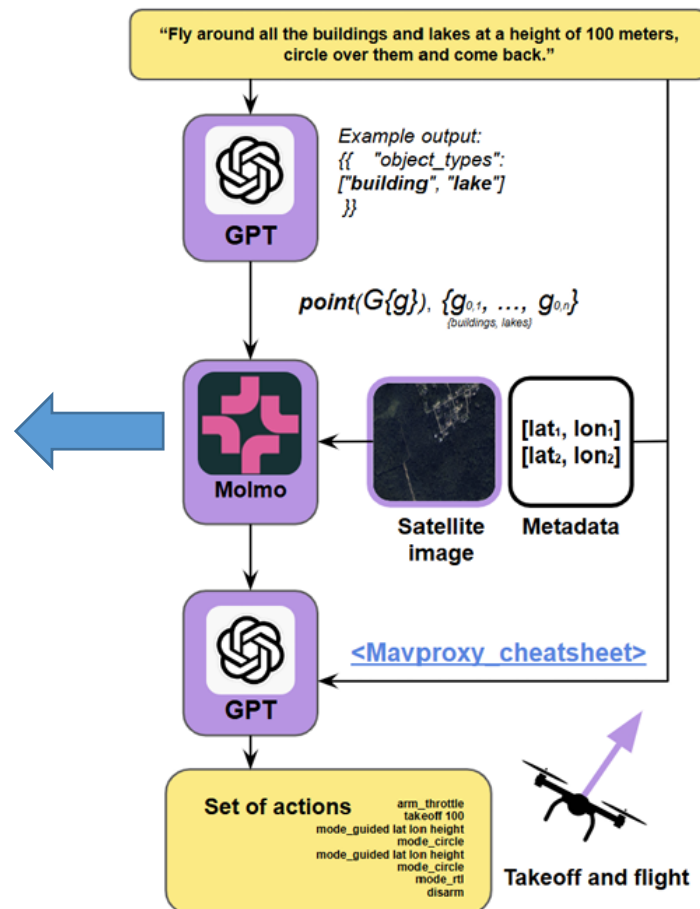
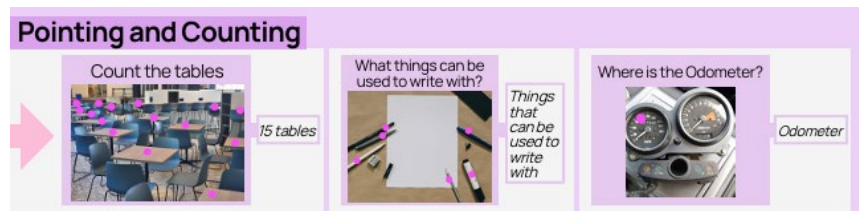
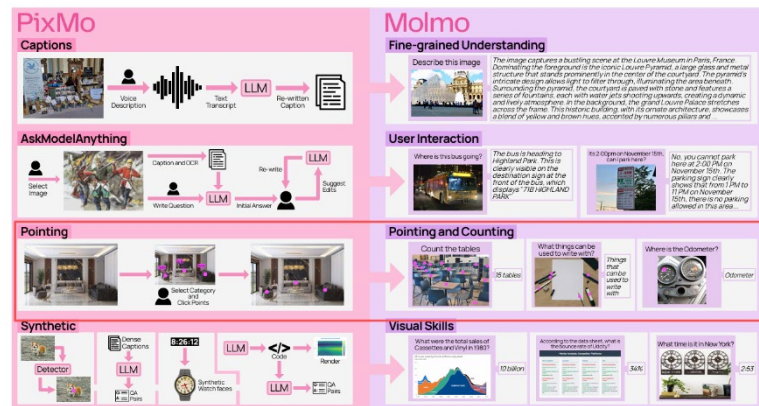
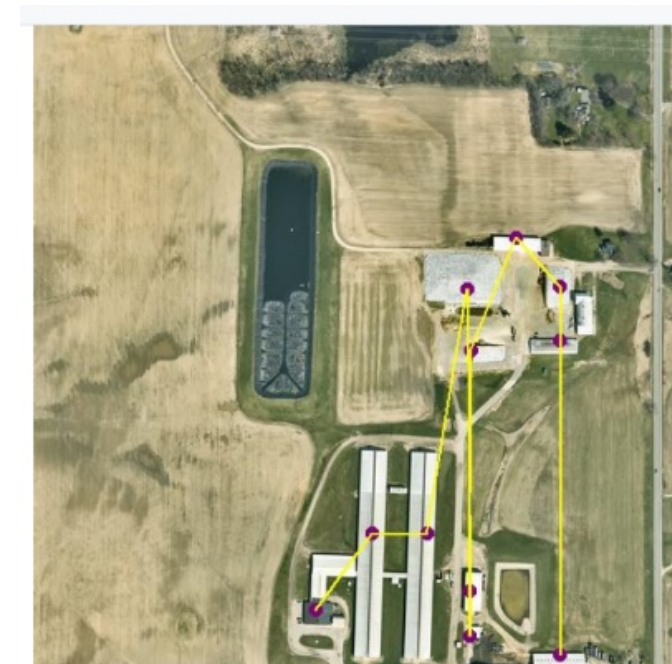
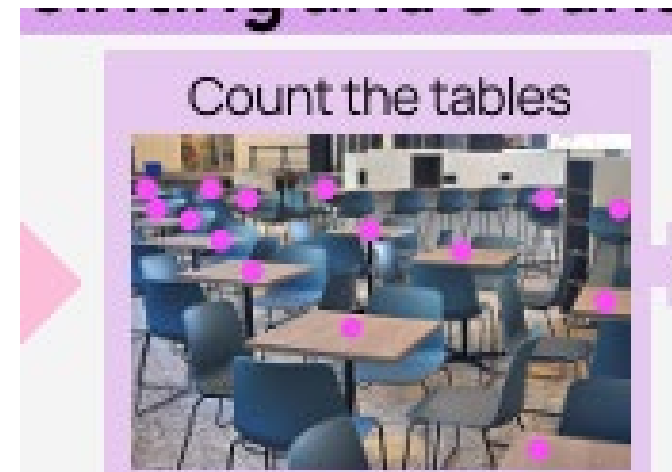


Fig. 1: The pipeline of the UAV-VLA system.



2

Method-步骤2.1-Molmo

输入: ->符合Molmo的数据格式

```
{
  "goals": [
    {
      "id": "A",
      "type": "start",
      "note": "home position (if known)"
    },
    {
      "id": "B",
      "type": "target",
      "action": "circle",
      "params": {
        "radius_m": 20,
        "alt_m": 100,
        "take_photo": true
      }
    },
    {
      "id": "C",
      "type": "target",
      "action": "go_to",
      "params": {
        "alt_m": 100
      }
    },
    {
      "id": "finish",
      "type": "return to launch"
    }
  ],
  "constraints": {
    "max_flight_time_min": 30,
    "no_fly_zones": []
  }
}
```

输出:

```
{
  "detections": [
    {
      "label": "B",
      "bbox": [2296, 1492, 2336, 1532],
      "centroid": [2316, 1512],
      "conf": 0.97
    },
    {
      "label": "C",
      "bbox": [320, 410, 360, 450],
      "centroid": [340, 430],
      "conf": 0.94
    },
    {
      "label": "A",
      "centroid": [50, 50],
      "conf": 0.99
    }
  ],
  "image_meta": {
    "width": 4096,
    "height": 4096,
    "crs": "EPSG:3857",
    "transform": [...]
  }
}
```

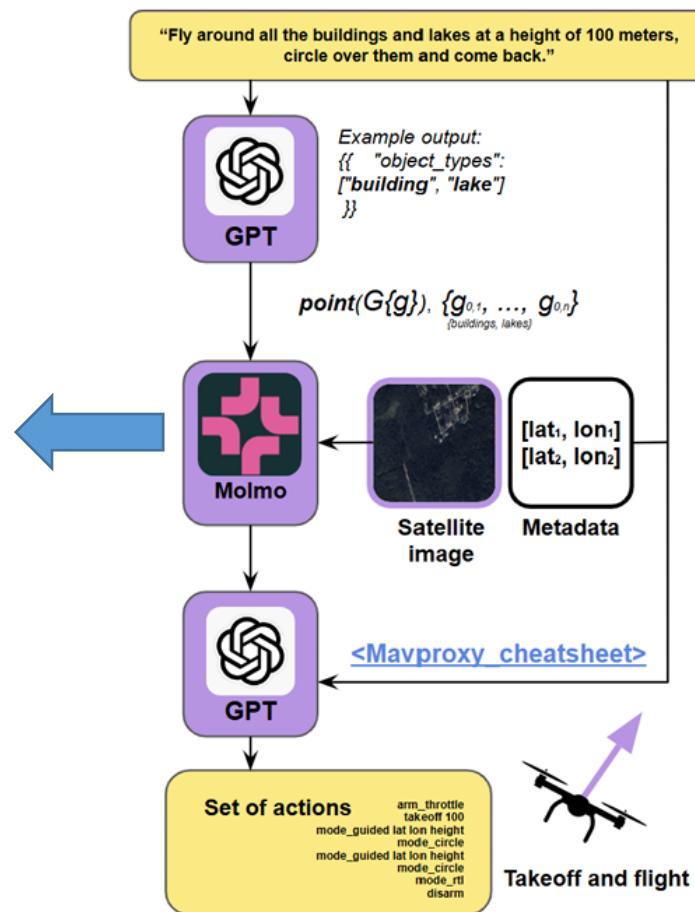
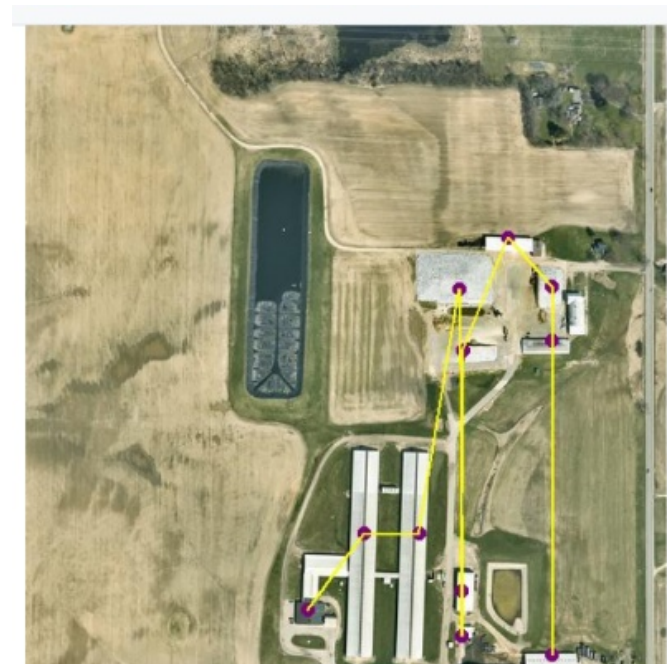
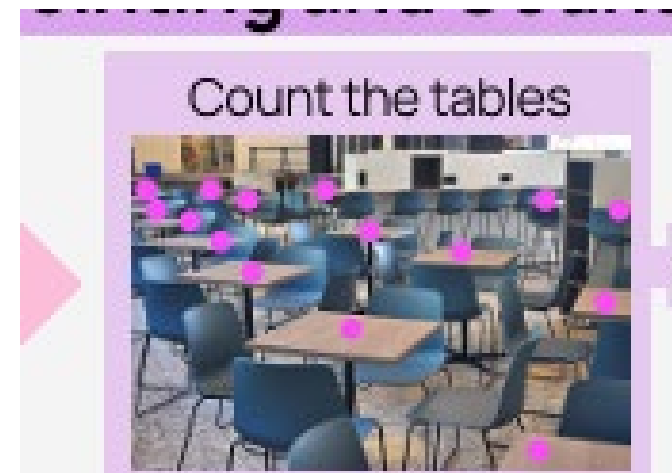


Fig. 1: The pipeline of the UAV-VLA system.



2

Method-步骤2.1-Molmo

注意:

加入指令中包含绕圈一周, 这种情况会将绕圈一周这种模拟数据转为离散数据, 比如给出8个点 (离散数据) 模拟绕圈一周的情况。

-这个**是Molmo模型的功能**。

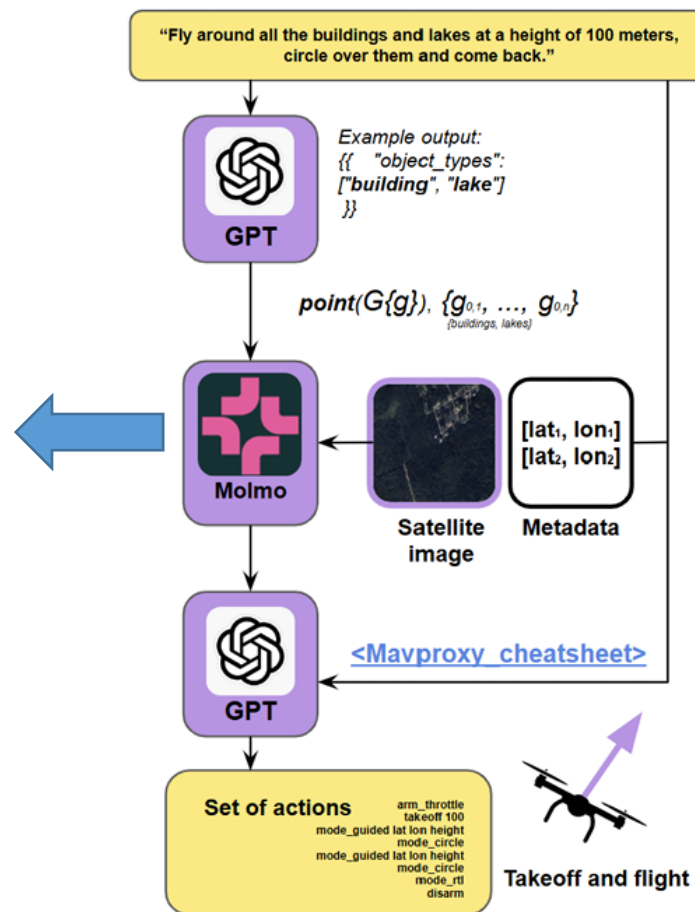
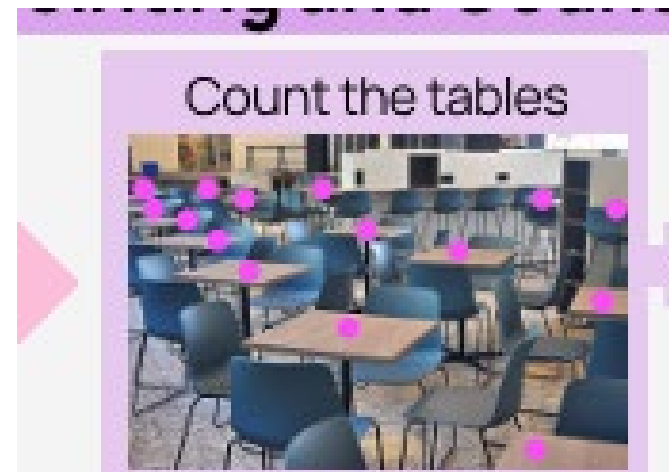


Fig. 1: The pipeline of the UAV-VLA system.



2

Method-步骤2.2-Metadata

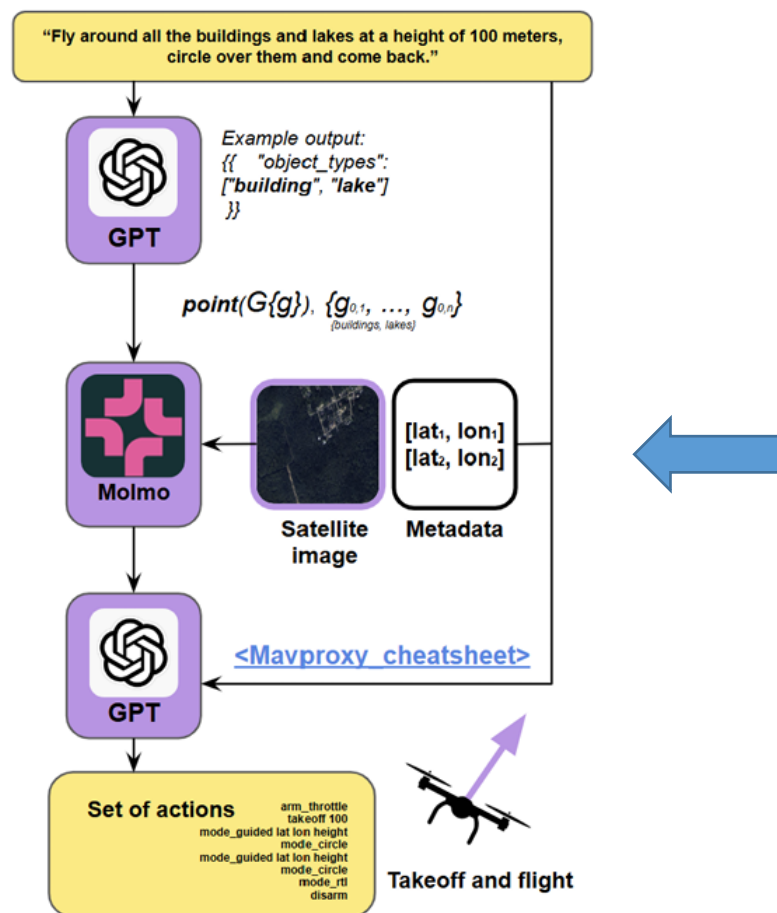


Fig. 1: The pipeline of the UAV-VLA system.

Metadata为卫星图像附带的地理空间信息。

这些信息一般包括：

- **地理参考信息 (Geo-referencing)**
 - 图像左上角（或某个角）的真实世界坐标（通常是经纬度或投影坐标系下的值）。
- **像素分辨率 (resolution)**
 - 每个像素对应的实际距离（比如 1.5 米/像素）。
- **投影信息 (Projection / CRS)**
 - 图像所用的坐标参考系统（如 WGS84 经纬度，或 UTM 投影坐标）。
- **仿射变换矩阵 (GeoTransform)**
 - 描述如何从像素坐标 (px, py) 映射到实际坐标 (X, Y)。

✎ **作用：**

有了这些 metadata，系统就能把 VLM 找到的像素点坐标 **投射到真实世界坐标**。
 比如：

$$X = X_{origin} + px \times pixel_size_x$$

$$Y = Y_{origin} - py \times pixel_size_y$$

然后再根据投影系统转为**经纬度 (lat, lon)**。

2

Method-步骤2.2-Metadata

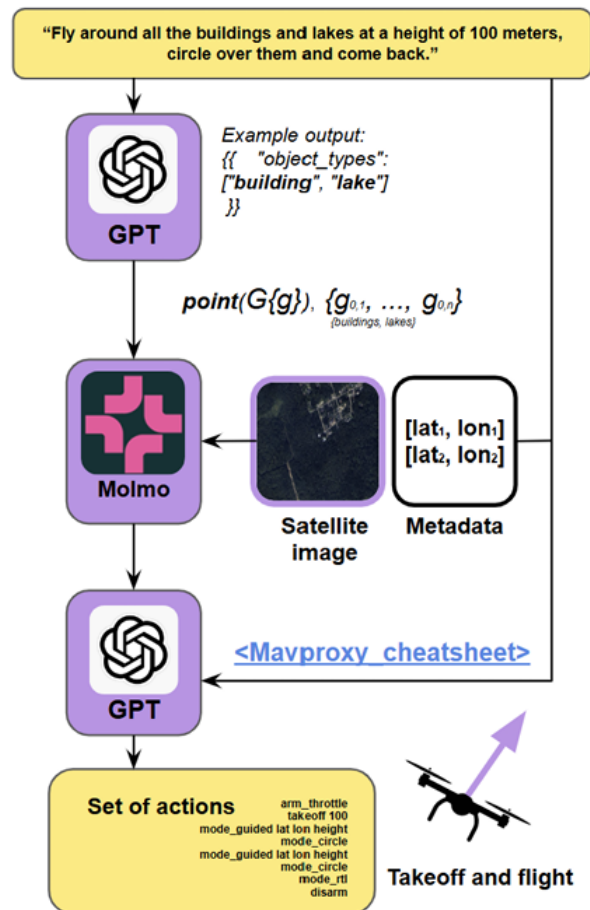


Fig. 1: The pipeline of the UAV-VLA system.

输入:

```
{ "detections": [
  {"label": "B", "bbox": [2296, 1492, 2336, 1532], "centroid": [2316, 1512], "conf": 0.97},
  {"label": "C", "bbox": [320, 410, 360, 450], "centroid": [340, 430], "conf": 0.94},
  {"label": "A", "centroid": [50, 50], "conf": 0.99}
],
"image_meta": {"width": 4096, "height": 4096, "crs": "EPSG:3857", "transform": [...]}}
```

输出:

```
{
  "targets_geo": [
    {"id": "A", "lat": 37.00050000, "lon": -122.00050000, "alt_m": 10},
    {"id": "B", "lat": 37.00100000, "lon": -122.00000000, "alt_m": 100},
    {"id": "C", "lat": 37.00000000, "lon": -121.99950000, "alt_m": 100}
  ]
}
```

Method-步骤3-GPT+Mavproxy cheatsheet

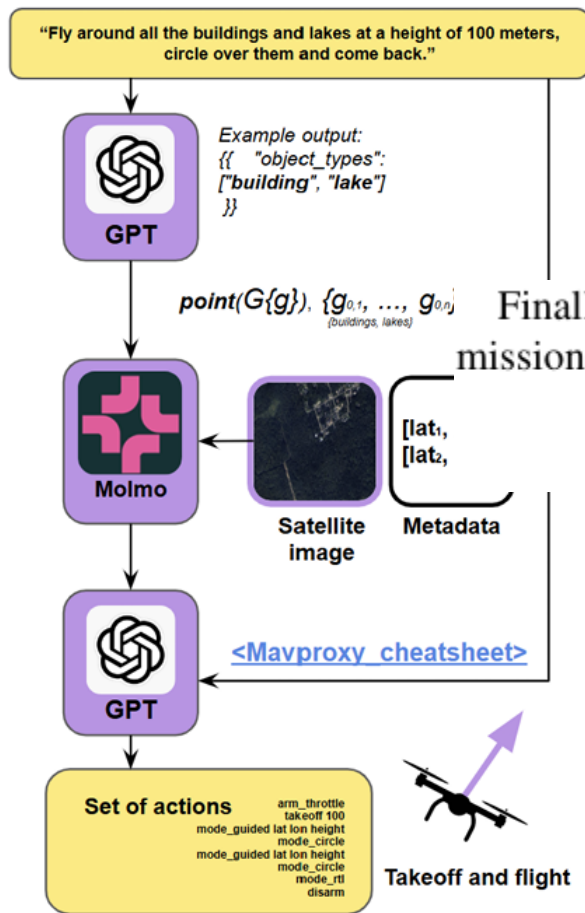


Fig. 1: The pipeline of the UAV-VLA system.

输入: 结构化目标信息+真实世界坐标+平台约束

- 结构化目标信息: $\{ \text{"detections": [$
 $\{ \text{"label": "B", "bbox": [2296, 1492, 2336, 1532], "centroid": [2316, 1512], "conf": 0.97},$
 $\{ \text{"label": "C", "bbox": [320, 410, 360, 450], "centroid": [340, 430], "conf": 0.94},$
 $\{ \text{"label": "A", "centroid": [50, 50], "conf": 0.99} \}],$

- 真实世界坐标: $\{ \text{"targets_geo": [$

Finally, the actions generation GPT module uses P_g , mission details, and MAVProxy [20] to generate UAV actions:

$$A = GPT(P_g, [A_b]) = \{A_1, A_2, \dots, A_n\} \quad (5)$$

- $\{ \text{"lat": 122.00050000, "alt_m": 10},$
 $\text{"lat": 122.00000000, "alt_m": 100},$
 $\text{"lat": 121.99950000, "alt_m": 100} \} \}$

some 点、飞行规则 (论文没提但应该加了)

输出:

```
{ "mission": [
  {"seq": 0, "cmd": "TAKEOFF", "lat": 37.0005, "lon": -122.0005, "alt_m": 10},
  {"seq": 1, "cmd": "WAYPOINT", "lat": 37.0010, "lon": -122.0000, "alt_m": 100, "speed_mps": 5},
  {"seq": 2, "cmd": "WAYPOINT", "lat": 37.0010, "lon": -121.99977503, "alt_m": 100},
  {"seq": 3, "cmd": "WAYPOINT", "lat": 37.00112704, "lon": -121.99984093, "alt_m": 100},
  ... (环绕 B 的其余航点) ...,
  {"seq": 10, "cmd": "WAYPOINT", "lat": 37.0000, "lon": -121.9995, "alt_m": 100}, // C 点
  {"seq": 11, "cmd": "RETURN_TO_LAUNCH"}
], "meta": { "generator": "ActionGenerationGPT_v1", "created": "2025-09-23T19:05:00+08:00" }}
```

Mavproxy是一个飞控接口工具, 通过指令、脚本实现无人机飞控

Method-步骤3-GPT+Mavproxy_cheatsheet

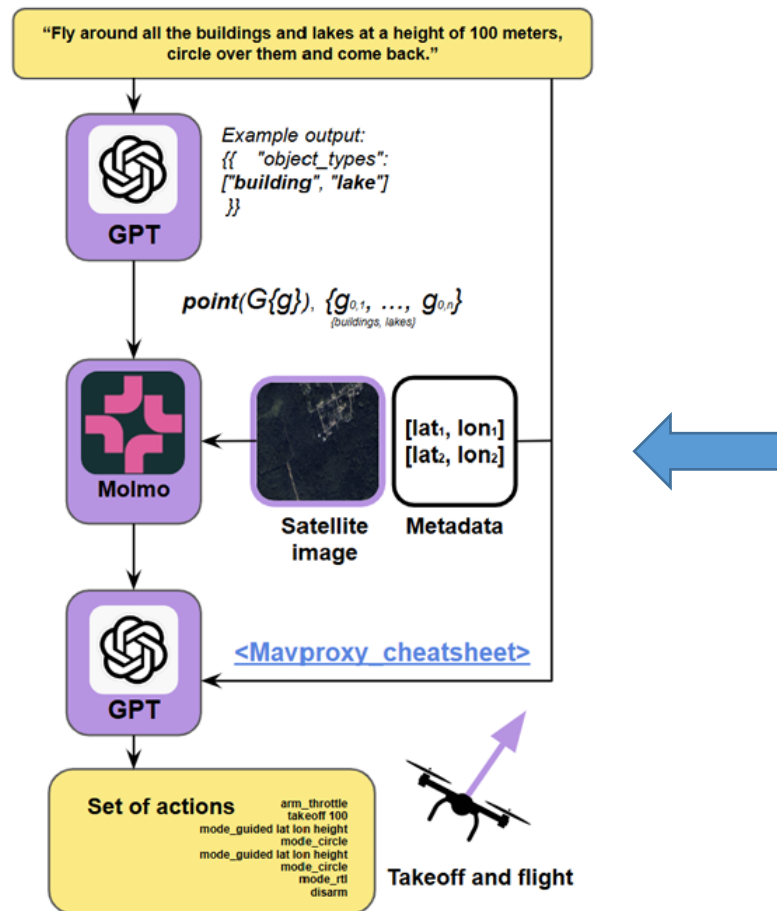


Fig. 1: The pipeline of the UAV-VLA system.

**最终输出的指令是MAVproxy_cheatsheet格式的
对应MAVProxy软件的控制命令**

MAVProxy cheatsheet 就是 MAVProxy 软件的命令速查表，方便快速查阅常用 UAV 控制命令。在论文里，它的作用是说明生成的动作序列可以直接映射成 MAVProxy 命令，从而让无人机执行任务。

Finally, the actions generation GPT module uses P_g , mission details, and MAVProxy [20] to generate UAV actions:

$$A = GPT(P_g, [A_b]) = \{A_1, A_2, \dots, A_n\} \quad (5)$$

具体怎么使用GPT和MAVProxy输出运动指令的，原文没有任何体现，需要从代码里查了，但代码数据目前没完全公开。

UPD from the authors: Check the new paper. The code and training data of the new paper will be released soon!

3

Experiments-设置与结果

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_n - \hat{x}_n)^2 + (y_n - \hat{y}_n)^2}, \quad (6)$$

“实验在配备RTX 4090显卡（24GB显存）和Intel Core i9-13900K处理器的个人电脑上进行。由于内存限制，使用了量化的Molmo-7B-D BnB 4位模型[22]。我们将无人机-VLA系统生成的飞行计划与人类生成的计划进行了比较。”



(a) Human-made



(b) UAV-VLA

TABLE I. COMPARISON OF RMSE METRICS FOR DIFFERENT METHODS

Metric (RMSE)	KNN (m)	DTW (m)	Sequential (m)
Mean	34.22	307.27	409.54
Median	26.05	318.46	395.59
Max	112.49	644.57	727.94

3

Experiments-指标

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_n - \hat{x}_n)^2 + (y_n - \hat{y}_n)^2}, \quad (6)$$



(a) Human-made



(b) UAV-VLA

1. Sequential Method (顺序对齐法)

- 思路：系统生成的轨迹点 按顺序 一一对应到人工轨迹点。
- 优点：实现简单，直观比较两条轨迹。
- 缺点：如果轨迹点数量或分布不一致，会出现误差累积，导致结果偏大。
- 类比：像是让两个人以相同的步伐走路，只要有一点步伐错位，后面就会越来越不一致。

2. DTW (Dynamic Time Warping, 动态时间规整)

- 思路：允许轨迹在时间维度上“拉伸/压缩”，通过灵活匹配点来最小化整体差异。
- 优点：能处理两条轨迹速度不同或点数不一致的情况，更关注“轨迹形状”是否相似。
- 缺点：计算复杂度比 Sequential 高。
- 类比：像是两个人走路时，一个快一个慢，DTW 会智能调整步子来匹配他们的路线，而不是要求严格同步。

3. KNN (K-Nearest Neighbors, 最近邻匹配)

- 思路：系统生成的每个点，都去找人工轨迹中最近的点来比较，完全忽略顺序。
- 优点：更关注空间位置的接近程度，误差通常最小。
- 缺点：不考虑路径的先后关系，可能忽视轨迹的顺序合理性。
- 类比：像是只比较两个人走过的“地点集合”，不关心他们是按什么顺序到的。

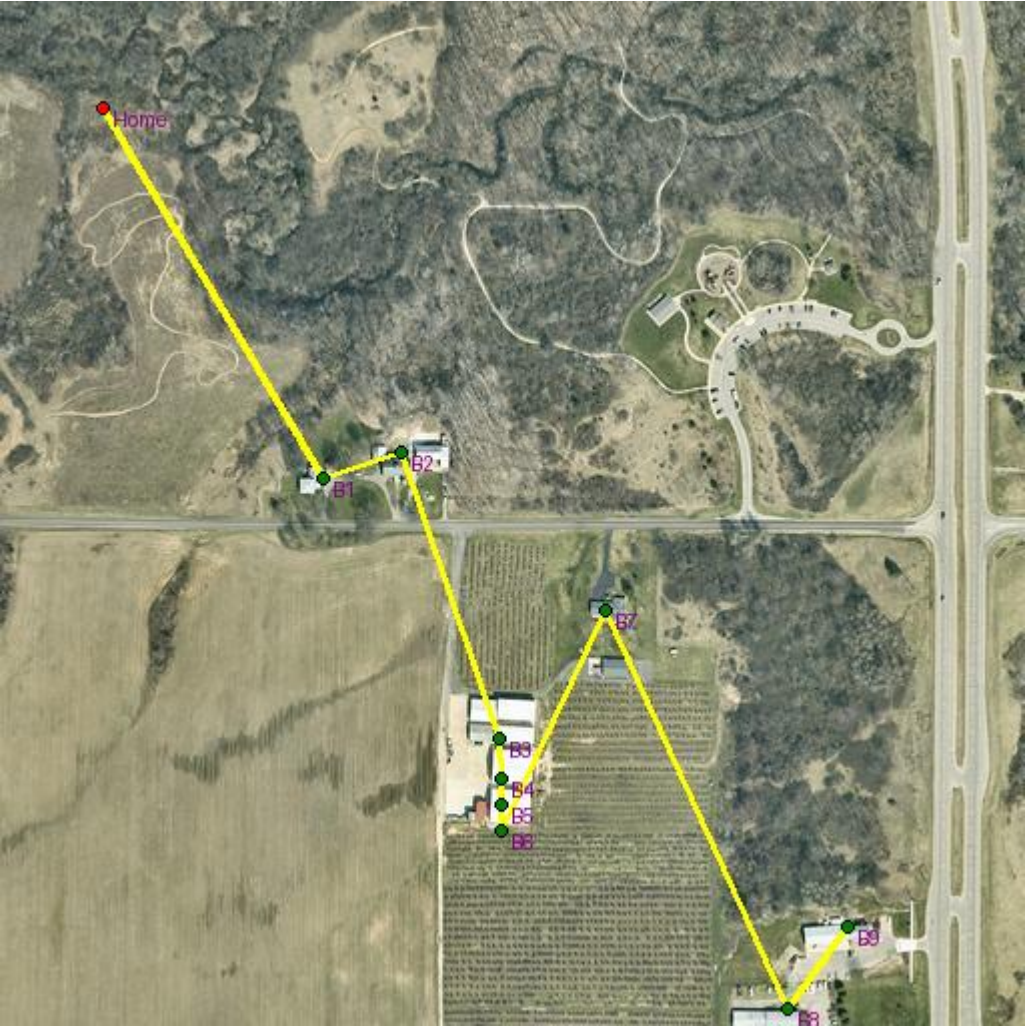
4

Results



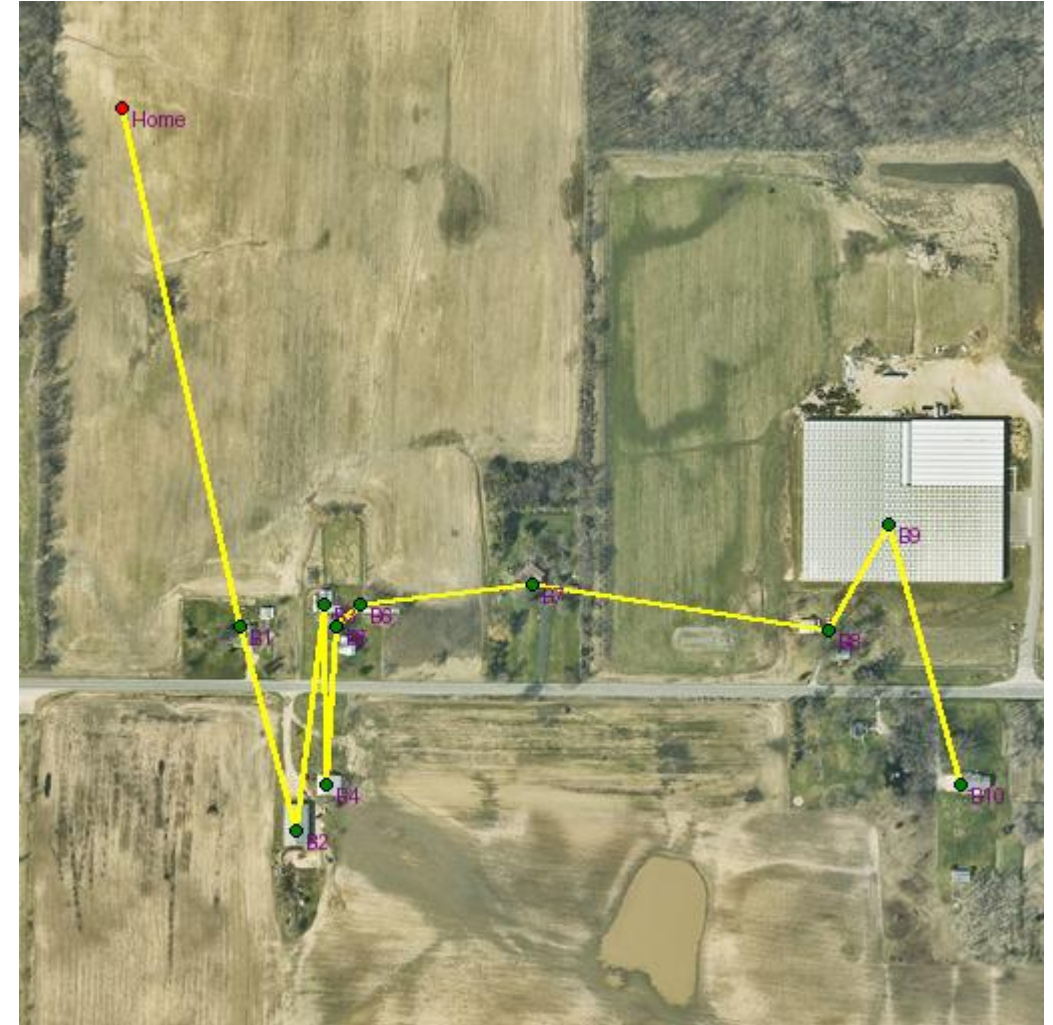
4

Results



4

Results



Thanks!

