四川大学

SICHUAN UNIVERSITY

2021年大创中期报告

项目名称：可移植视觉综合珍稀动物实时检测与跟踪系统

项目人员：冯云龙 李昊 王雅璇 任浩龙 钟程澜

指导教师：赵启军

2021 年 5 月 2 日

目 录

第一章 项目主要进展

1.1 概述

我国是珍稀动物大国，据不完全统计，仅列入《濒危野生动植物种国际贸易公约》附录的原产于中国的濒危动物就有 120 多种，物种多样性急需保护。但是物种监测站布局有限，难以全面覆盖野生动物集中活动区域，还存在监测盲区，监测能力还有待提高，一些技术问题尚未解决。传统野生动物保护监测方式不利于研究人员全面、准确、及时地掌握野生动物资源现状以及野生动物资源的动态变化。

本项目致力于设计用于科考珍稀动物的全天候摄像追踪系统，为野外生态科考提供动物图像数据，进而为生态科研提供相关的支持，帮助生态研究人员全面、准确、及时地掌握野生动物资源现状及动态变化。目前的科考摄像主要采用红外传感定点定角度拍摄，拍摄到的图片和视频具有局限性。而本项目使用**云台摄像头，检测并锁定目标物种，根据对象的实时相对位置来修正云台摄像头角度姿态参数，保证对目标锁定追踪获得更好的拍摄效果**来辅助野外科考。

除此之外，本项目开发的在线网页“西藏动物追踪平台”，可以给广大动物保护者、爱好者，藏区旅游者，动物保护科研人员，生态学家等提供一个**在线实时更新的动物检测识别与跟踪平台**，从数据大屏直观的了解到藏区珍稀动物与危险动物实时分布，进而去调整旅行计划与相应的科研计划。

本项目的动物摄像追踪系统具有以下特征：

- **可移植性强**，能在不同的运载平台上完成识别和追踪任务；
- **云台全方位跟踪拍摄**，弥补时下生态科考中拍摄视角固定，拍摄效率低下，拍摄成本较高的缺陷；

- **高鲁棒性的目标检测与自动跟踪技术**，应用时下效果最好的深度学习方法，训练出具有高鲁棒性、流畅性的算法模型，对运动中的动物进行运动建模，并预测下一时间段出现的坐标，具有更好的识别检测与自动跟踪效果。

到目前为止，团队已经完成**云台系统结构的设计和组装、云台控制算法、动物检测与识别模块、动物自动追踪模块、Web 展示平台的建立、数据集的初步建立**等任务，项目成果成熟度较高，在现有的珍稀动物数据集上也取得了较好的识别和跟踪效果，在实际野外环境中将更具实用价值。

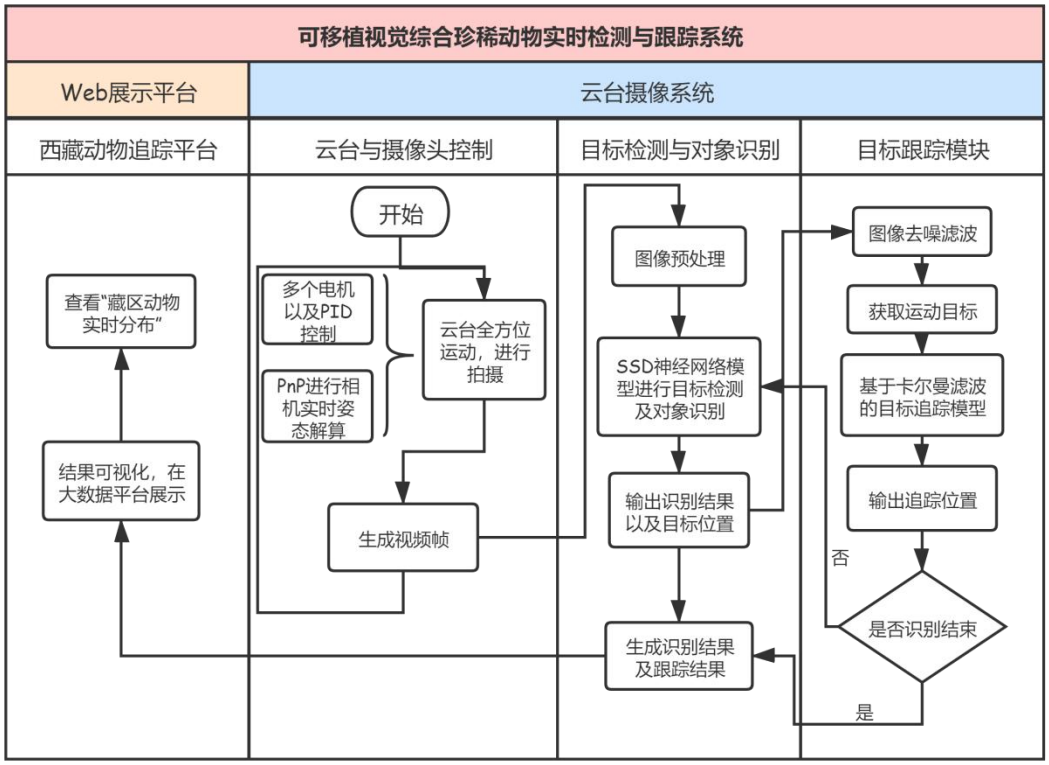


图 1.1 项目主要进度展示

1.2 云台摄像系统设计

云台作为赋予摄像系统转向能力的部分，需要保证稳定性，运动的敏捷性，足够大的转向角度，以实现精确的跟踪定位。

目前我们的云台摄像系统已经实现了以下功能：

- **快速拆卸结构：**方便把摄像系统在多种环境之中固定和拆卸，为日常维护与障碍维修提供了极大便利。

- **360°全方位云台摄像头：**使用多个电机控制云台，使得云台能够全方位的定位目标得到更好的拍摄效果。

- **云台控制算法流畅稳定：**云台控制模块需要根据摄像模块实时解算的姿态来控制 yaw 和 pitch 轴，达到固定追踪的目的。我们使用了 PID（比例积分微分控制算法）和卡尔曼滤波算法，以保证云台和摄像模块有稳定良好的通讯和机械结构控制的稳定性。

- **相机姿态解算：**摄像模块可以根据相机相对于目标动物在世界坐标系下的相对位移和旋转矩阵来解算相机的目标姿态，并结合云台的控制模块调整相机的角度锁定目标动物。

1.2.1 云台摄像系统机械结构设计

因野外携带需要，在设计中我们尽可能的进行轻量化设计以减轻云台与摄像机的重量。云台摄像系统整体移植性较好，可以方便的实现在多种运载设备上的安装。

云台以及双目摄像头的整体设计图纸如下所示：

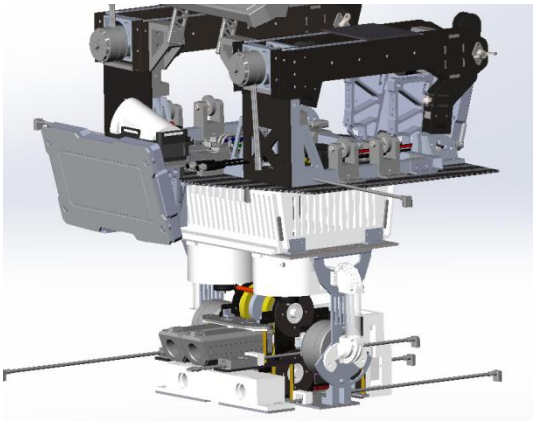


图 1.2　云台及摄像头图纸

云台机械结构设计实物如下图：

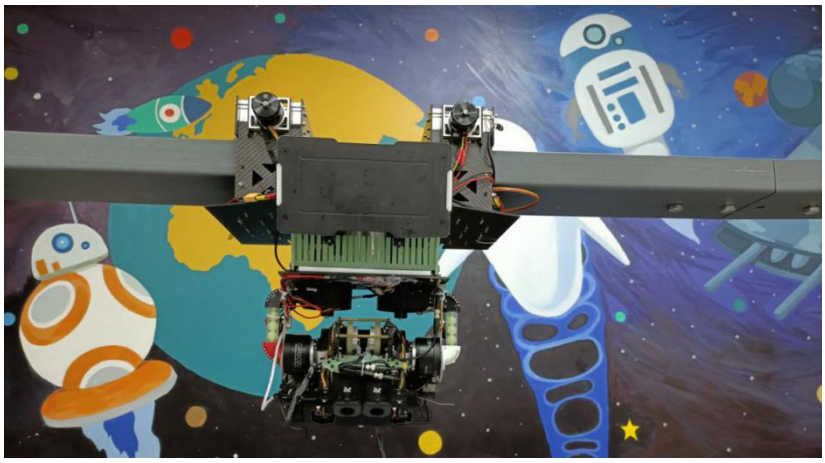


图 1.3　云台摄像系统实物图

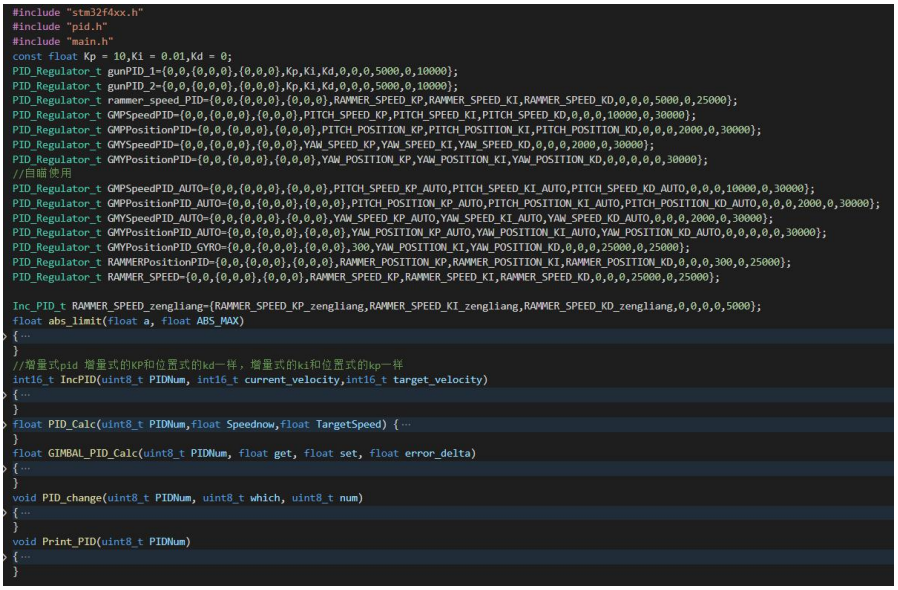
1.2.2 云台控制算法

为了使云台能全方位拍摄检测、识别和跟踪目标，我们采取了多电机控制的策略，使得云台可以全方位的转动。并且为了达到固定追踪的目的，需要根据摄像模块实时解算的姿态来控制 yaw 和 pitch 轴。我们使用了 PID（比例积分微分控制算法）和卡尔曼滤波算法，以保证云台和摄像模块有稳定良好的通讯和机械结构控制的稳定性。其中卡尔曼滤波算法将在 1.4.X 中进行介绍。

PID（比例积分微分控制算法）就是根据系统的误差,利用比例、积分、微分计算出控制量进行控制的。受控变数是三种算法（比例、积分、微分）相加后的结果，即为其输出，其输入为误差值（设定值减去测量值后的结果）或是由误差值衍生的信号。其输入输出的关系如下所示：

$$\mathrm{u}(t) = \mathrm{MV}(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{d}{dt} e(t)$$

我们利用 PID 控制算法，实现了对云台的控制，保证了机械结构控制的稳定性。PID 控制部分代码如下：



```
#include "stm32f4xx.h"
#include "pid.h"
#include "main.h"
const float Kp = 10,Ki = 0.01,Kd = 0;
PID_Regulator_t gunPID_1={0,0,{0,0,0},{0,0,0},Kp,Ki,Kd,0,0,0,5000,0,10000};
PID_Regulator_t gunPID_2={0,0,{0,0,0},{0,0,0},Kp,Ki,Kd,0,0,0,5000,0,10000};
PID_Regulator_t rammer_speed_PID={0,0,{0,0,0},{0,0,0},RAMMER_SPEED_KP,RAMMER_SPEED_KI,RAMMER_SPEED_KD,0,0,0,5000,0,25000};
PID_Regulator_t GMPSpeedPID={0,0,{0,0,0},{0,0,0},PITCH_SPEED_KP,PITCH_SPEED_KI,PITCH_SPEED_KD,0,0,0,10000,0,30000};
PID_Regulator_t GMPPositionPID={0,0,{0,0,0},{0,0,0},PITCH_POSITION_KP,PITCH_POSITION_KI,PITCH_POSITION_KD,0,0,0,2000,0,30000};
PID_Regulator_t GMYSpeedPID={0,0,{0,0,0},{0,0,0},YAW_SPEED_KP,YAW_SPEED_KI,YAW_SPEED_KD,0,0,0,2000,0,30000};
PID_Regulator_t GMYPositionPID={0,0,{0,0,0},{0,0,0},YAW_POSITION_KP,YAW_POSITION_KI,YAW_POSITION_KD,0,0,0,0,0,30000};
//自瞄使用
PID_Regulator_t GMPSpeedPID_AUTO={0,0,{0,0,0},{0,0,0},PITCH_SPEED_KP_AUTO,PITCH_SPEED_KI_AUTO,PITCH_SPEED_KD_AUTO,0,0,0,10000,0,30000};
PID_Regulator_t GMPPositionPID_AUTO={0,0,{0,0,0},{0,0,0},PITCH_POSITION_KP_AUTO,PITCH_POSITION_KI_AUTO,PITCH_POSITION_KD_AUTO,0,0,0,2000,0,30000};
PID_Regulator_t GMYSpeedPID_AUTO={0,0,{0,0,0},{0,0,0},YAW_SPEED_KP_AUTO,YAW_SPEED_KI_AUTO,YAW_SPEED_KD_AUTO,0,0,0,2000,0,30000};
PID_Regulator_t GMYPositionPID_AUTO={0,0,{0,0,0},{0,0,0},YAW_POSITION_KP_AUTO,YAW_POSITION_KI_AUTO,YAW_POSITION_KD_AUTO,0,0,0,0,0,30000};
PID_Regulator_t GMYPositionPID_GYRO={0,0,{0,0,0},{0,0,0},300,YAW_POSITION_KI,YAW_POSITION_KD,0,0,0,25000,0,25000};
PID_Regulator_t RAMMERPositionPID={0,0,{0,0,0},{0,0,0},RAMMER_POSITION_KP,RAMMER_POSITION_KI,RAMMER_POSITION_KD,0,0,0,300,0,25000};
PID_Regulator_t RAMMER_SPEED={0,0,{0,0,0},{0,0,0},RAMMER_SPEED_KP,RAMMER_SPEED_KI,RAMMER_SPEED_KD,0,0,0,25000,0,25000};

Inc_PID_t RAMMER_SPEED_zengliang={RAMMER_SPEED_KP_zengliang,RAMMER_SPEED_KI_zengliang,RAMMER_SPEED_KD_zengliang,0,0,0,0,5000};
float abs_limit(float a, float ABS_MAX)
{…
}
//增量式pid 增量式的KP和位置式的kd一样，增量式的ki和位置式的kp一样
int16_t IncPID(uint8_t PIDNum, int16_t current_velocity,int16_t target_velocity)
{…
}
float PID_Calc(uint8_t PIDNum,float Speednow,float TargetSpeed) {…
}
float GIMBAL_PID_Calc(uint8_t PIDNum, float get, float set, float error_delta)
{…
}
void PID_change(uint8_t PIDNum, uint8_t which, uint8_t num)
{…
}
void Print_PID(uint8_t PIDNum)
{…
}
```

```
if(state == 1)
{
    if(last_state!=1)
    {
        FrictionRamp1.ResetCounter(&FrictionRamp1);
        FrictionRamp2.ResetCounter(&FrictionRamp2);
    }
    current1=PID_Calc(1,fric_data1.speed,targetspeed1*FrictionRamp1.Calc(&FrictionRamp1));
    SetMotorCurrent(FRIC1_MOTOR, current1);
    current2=PID_Calc(2,fric_data2.speed,targetspeed2*FrictionRamp2.Calc(&FrictionRamp2));
    SetMotorCurrent(FRIC2_MOTOR, current2);
}
if(state != 1)
{
    if(last_state==1)
    {
        FrictionRamp1.ResetCounter(&FrictionRamp1);
        FrictionRamp2.ResetCounter(&FrictionRamp2);
    }
    current1=PID_Calc(1,fric_data1.speed,fric_data1.speed*(1.0f-FrictionRamp1.Calc(&FrictionRamp1)));
    SetMotorCurrent(FRIC1_MOTOR, current1);
    current2=PID_Calc(2,fric_data2.speed,-fric_data1.speed*(1.0f-FrictionRamp2.Calc(&FrictionRamp2)));
    SetMotorCurrent(FRIC2_MOTOR, current2);
}
SendMotorCurrent(FRIC1_MOTOR);
SendMotorCurrent(FRIC2_MOTOR);
//  printf("v:%d,%d,%d\n",fric_data1.speed,targetspeed1,current1);
last_state = state;
```

图 1.4 PID 云台控制部分代码

1.2.3 相机姿态解算

姿态解算是摄像头云台控制的基础、重要部分，估计出来的姿态会发布给姿态控制器，控制云台 yaw 轴 pitch 轴转向，进而达到固定追踪的目的。我们使用了 PnP 求解算法来计算相机相对于目标动物在世界坐标系下的相对位移和旋转矩阵，进而解算相机的目标姿态，再结合云台的控制模块调整相机的角度锁定目标动物。

通过三维坐标计算欧拉角，进而解算云台的 yaw 轴和 pitch 轴的代码如下：

```
def transform(self, position, v):
    """
TODO 确认单位，正负
:param position: 相机坐标系中目标的位置，单位 cm
:param v: 初速度，单位 m/s
:return: pitch轴和yaw轴的转角，单位 度
"""
pitch_rad = self.getPitch((position[2] + self.offset_x) / 100, -(position[1] + self.offset_y) / 100, v)
pitch = (pitch_rad / np.pi * 180) + self.offset_pitch

# yaw = -(np.arctan2(position[0] + self.offset_x, position[2] + self.offset_z)) + self.offest_yaw
yaw_rad = np.arctan2(position[0] + self.offset_x, position[2] + self.offset_z)
yaw = (yaw_rad / np.pi * 180) + self.offset_yaw

return pitch, yaw
```

图 1.5 解算 yaw 轴和 pitch 轴代码

1.3 动物目标检测和物种识别

本项目的**目标检测和识别模型**是部署在可移动的嵌入式芯片上的，且主要采用计算速度快、模型参数少、效果较好的 SSD 模型对目标进行识别。

1.3.1 数据获取及预处理

我们将现有的无人机俯拍视角的动物运动数据集进行收集整理，并利用 labelMe 工具进行标记，将标记好的数据集的 80%用于模型训练，10%作为验证集，剩下的 10%作为测试集。

图 1.6 标记好的数据集部分展示

为了解决图像获取过程中产生的噪点和干扰数据，我们通过中值、高斯以及均值滤波算法去除线性加法噪声，通过傅里叶变换去除乘法噪声。

1.3.2 SSD 模型训练

考虑到移动嵌入式平台的计算能力，我们主要采用 SSD 模型进行目标检测和对象识别。SSD 是一种 one-stage 的通用物体检测算法，适合在移动嵌入式平台部署。

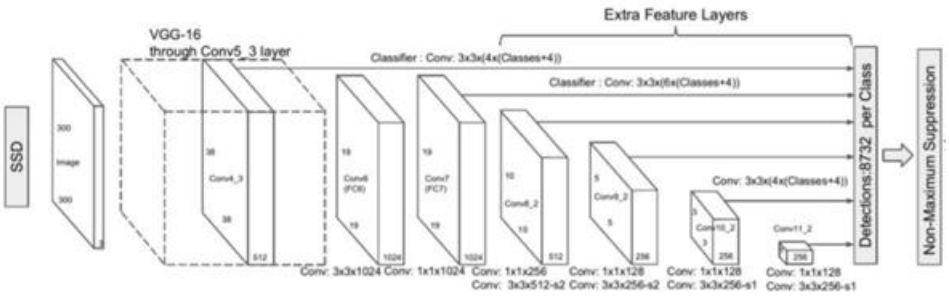


图 1.7 SSD 网络架构

SSD 模型的部分代码如下所示：

```
class SSD(BaseArch):
    __category__ = 'architecture'
    __inject__ = ['post_process']
    def __init__(self, backbone, ssd_head, post_process):...
    @classmethod
    def from_config(cls, cfg, *args, **kwargs):...
    def _forward(self):
        # Backbone
        body_feats = self.backbone(self.inputs)
        # SSD Head
        if self.training:
            return self.ssd_head(body_feats, self.inputs['image'],
                                 self.inputs['gt_bbox'],
                                 self.inputs['gt_class'])
        else:
            preds, anchors = self.ssd_head(body_feats, self.inputs['image'])
            bbox, bbox_num = self.post_process(preds, anchors,
                                               self.inputs['im_shape'],
                                               self.inputs['scale_factor'])
            return bbox, bbox_num
    def get_loss(self, ):...
    def get_pred(self):...
```

动物检测和识别效果如下所示：

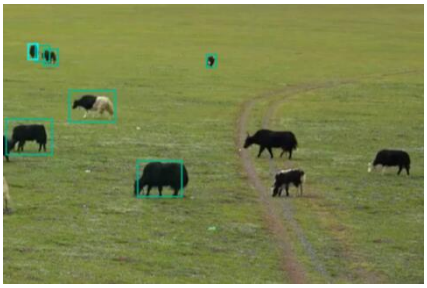


图 1.8 动物检测和识别效果图

1.3.3 数据归约处理

由于输入的数据存在噪声以及过拟合的情况，SSD 模型输出的结果可能会包含部分错误识别的结果，通常为 FP 类型的错误，所以我们采用了多帧拟合的方法来除去 FP 结果，使得结果更具可信度。

具体方法为：若一个对象在相邻的若干帧中都未被识别，但在此帧中被检出，需要根据置信度进行筛选；若干帧中某一帧未被检出，则通过相邻数据 MEMC 算法进行补帧。

1.4 目标物种跟踪拍摄模块

在多种自动摄像系统中缺乏自动跟踪功能，目前这种功能只应用于人脸跟踪，并未扩展到其他领域。野生动物具有高灵活性，运动速度快且反应灵敏。目前已有的人脸追踪算法如果直接应用在动物上有较大的难度，于是我们团队开发出了一种新型的对于高速运动鲁棒性高的跟踪方法。

本模块利用无损卡尔曼滤波技术，对高速运动的动物建立运动模型，以实现对动物运动的预测，进而实现鲁棒性好、性能高的追踪算法。

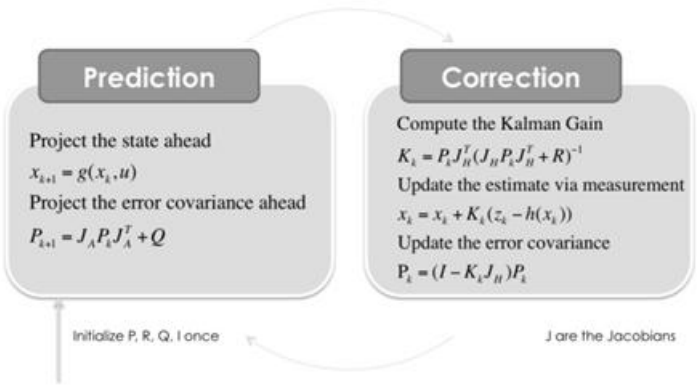


图 1.9 扩展卡尔曼滤波的全过程

并且，本项目中主要使用了恒定速度模型 CV 和恒定转率速度模型 CTRV 两种运动模型来进行测试和部署。

1.4.1 无损卡尔曼滤波实现

卡尔曼滤波算法是根据运动模型的预测值与观测值来自动计算预测对象位置的概率分布方法。由于动物的运动模型为非线性模型，无法使用普通的卡尔曼滤波，故我们使用了能够解决非线性问题的滤波器 EFK(扩展卡尔曼滤波)。具体算法实现如下：

```
class KalmanFilter(object):
    def __init__(self):…

    def track(self,x,y,z):
        self.last_prediction = self.current_prediction # 把当前预测存储为上一次预测
        self.last_measurement = self.current_measurement # 把当前测量存储为上一次测量

        if self.lostflag == 0:              #有目标的情况下丢目标
            if (abs(self.last_measurement[0] - x) > 15 or abs(self.last_measurement[1] - y) > 15 or abs(self.last_measurement[2] - z) > 15): #步兵移动速度3.5，一帧0.04s
                self.error_frame = self.error_frame + 1
            else:
                self.error_frame = 0

            if self.error_frame > 0 and self.error_frame < self.error_frame_size:
                self.current_measurement = np.array([[np.float32(self.last_prediction[0])],
                                                     [np.float32(self.last_prediction[1])],
                                                     [np.float32(self.last_prediction[2])]])
            elif self.error_frame >= self.error_frame_size:
                self.current_measurement = np.array([[np.float32(x)], [np.float32(y)], [np.float32(z)]]) # 当前测量
                self.lostflag = 1
                self.error_frame = 0
            elif  self.error_frame == 0:
                self.current_measurement = np.array([[np.float32(x)], [np.float32(y)], [np.float32(z)]])  # 当前测量
                self.lostflag = 0
        if self.lostflag == 1:
            self.current_measurement = np.array([[np.float32(x)], [np.float32(y)], [np.float32(z)]])
        self.kalman.correct(self.current_measurement) # 用当前测量来校正卡尔曼滤波器
        self.current_prediction = self.kalman.predict() # 计算卡尔曼预测值，作为当前预测
        #lmx, lmy, lmz = self.last_measurement[0], self.last_measurement[1], self.last_measurement[2] # 上一次测量坐标
        cmx, cmy, cmz = self.current_measurement[0], self.current_measurement[1], self.current_measurement[2] # 当前测量坐标
        #lpx, lpy, lpz = self.last_prediction[0], self.last_prediction[1], self.last_prediction[2] # 上一次预测坐标
        cpx, cpy, cpz = self.current_prediction[0], self.current_prediction[1], self.current_prediction[2] # 当前预测坐标
        if self.lostflag == 1:
            self.counter = self.counter - 1
            if(self.counter == 0):
                self.lostflag = 0
                self.counter = self.get_target_size
            return cmx,cmy,cmz
        else:
            return cpx,cpy,cpz
```

算法 1: 卡尔曼滤波算法

输入： 输入一个 object 在多维坐标系下的坐标

输出: 该 object 在下一时刻的坐标

for loop←1 **to** 最大迭代数 **do**

通过卡尔曼滤波的五个方程，整合预测值及实际值，修正滤波器，得到在协方差最小情况下的预测坐标（如果当前观测值突变为 0，则以上一次的预测值作为本次观测值）

end

1.4.2 参数设置

参数	值		
时间序列错误帧窗口大小	6		
重新寻找目标缓存时间	4		
系统测量矩阵	$\begin{bmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\end{bmatrix}$		
状态转移矩阵	CV	CVA	CTRV
	$\begin{bmatrix}1&0&0&dt&0&0\\0&1&0&0&dt&0\\0&0&1&0&0&dt\\0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&0&1\end{bmatrix}$	$\begin{bmatrix}1&0&0&dt&0&0&0&0&0&0\\0&1&0&0&dt&0&0&0&0&0\\0&0&1&0&0&dt&0&0&0&0\\0&0&0&1&0&0&dt&0&0&0\\0&0&0&0&1&0&0&dt&0&0\\0&0&0&0&0&1&0&0&dt&0\\0&0&0&0&0&0&1&0&0&0\\0&0&0&0&0&0&0&1&0&0\\0&0&0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&0&0&0\end{bmatrix}$	
系统过程噪声协方差	$\begin{bmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&0&1\end{bmatrix}$		

表 1.1 卡尔曼滤波算法参数设置

1.4.3 基于卡尔曼滤波的跟踪模型

目标跟踪模型的具体步骤如下所示：

① 对一个连续的视频帧序列进行处理；

② 针对每一个视频序列，利用目标检测模块，检测可能出现目标；

③ 如果某一帧出现了前景目标，找到其具有代表性的关键特征点；

④ 对之后的任意两个相邻视频帧而言，寻找上一帧中出现的关键特征点；在当前帧中的最佳位置，从而得到前景目标在当前帧中的位置坐标；

⑤ 如此迭代进行，实现目标的跟踪。

具体算法实现如下：

```python
class track_3D:
    '''
    3D 目标跟踪预测
    '''
    def __init__(self, method="CV"):
        '''3D 目标检测
        使用无损卡尔曼滤波单目标跟踪/预测
        Parameters:
            method: str: 3D 运动模型. "CV"/"CVA"/"CTRV"
        '''
        if not (method in ["CV", "CVA", "CTRV"]):
            raise ValueError("Method should in \"CV\"/\"CVA\"/\"CTRV\"")
        self.method = method
        if method == "CVA": ···
        elif method == "CTRV": ···
        else: ···
        self.time_stamp = time.time()   # 预测时间戳
    def predict(self, dt=None):
        '''
        预测运动轨迹
        Parameters:
            dt: float
                预测dt(s)时间后的状态
        '''
        if dt is None:
            dt = time.time() - self.time_stamp
        self.kf.predict(dt=dt)
        self.time_stamp = time.time()
        return self.kf.x[:3]
    def update(self, z):
        '''
        使用观测值更新滤波器
        Parameters:
            z: numpy.array
                3维空间坐标(x, y, z)
        '''
        self.kf.update(z)
```

图 1.10 目标跟踪模型

目标跟踪结果如下所示：

图 1.11 动物目标跟踪图（左右为同一视频不同时间的跟踪效果）

1.5 Web展示平台

本项目的在线网页展示平台名为“西藏动物追踪平台”。本项目的动物云台摄像系统将被部署在藏区的各个野生动物出没点，进行全天候的自动检测识别与追踪，而收集到的信息经过系统处理加工，生成对应的统计数据到数据库中，再显示到“西藏动物追踪平台”，供广大科研工作者和动物爱好保护者进行实时查看和研究。

1.5.1 网站门户

本页面采用纯 JS 结合 Echarts 的方案，主要包括平台介绍、藏区野生动物种类及分布、藏区动物实时分布、危险动物预警等多个模块。

同时我们网页已经部署到服务器后端环境及数据传输还在不断完善（https://animal-tracking-tibet.99pika.com/，点击链接即可进入网站）

平台部分功能模块展示如下：

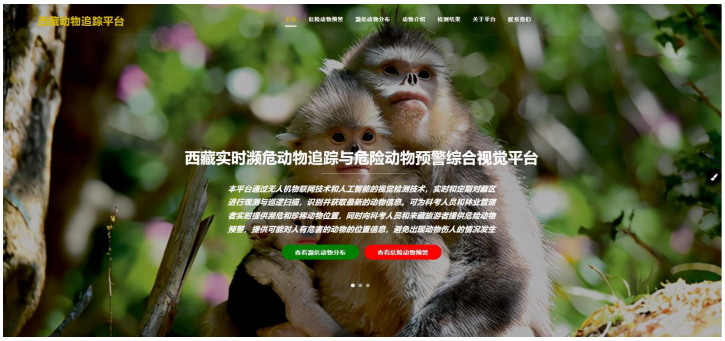


图 1.12 网站首页

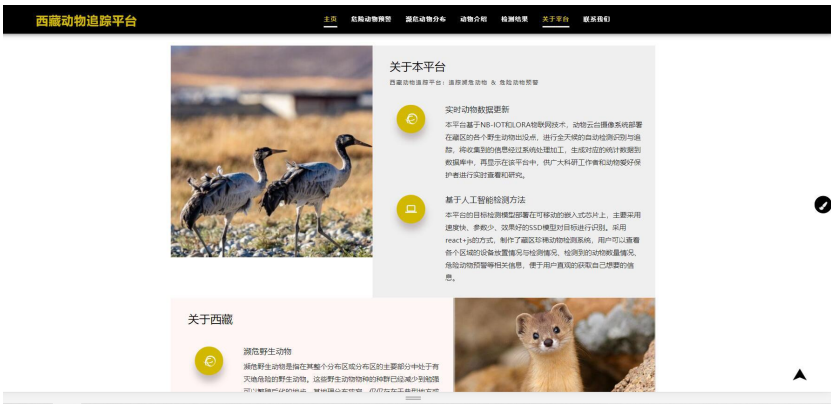


图 1.13 平台的基本介绍

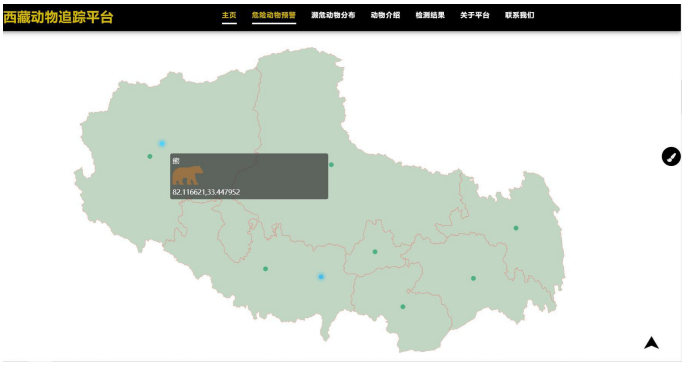


图 1.14 藏区动物分布和实时监测

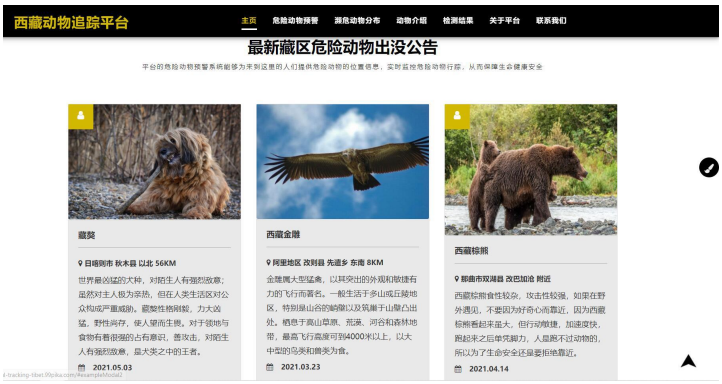


图 1.15 平台消息广播和资讯传递

图 1.16 检测成果展示

1.5.2 藏区动物检测系统

采用 react 技术栈+Echarts+DataV 数据展示的方式，制作了藏区珍稀动物监测系统。用户可以查看各个区域的设备放置情况与检测情况、检测到的动物数量情况、危险动物预警以及珍稀动物数量等相关信息，便于管理者直观的获取自己想要的信息。

网页整体采用 react 栈构建保证开发可重用性，理论上可以快速根据自己地区位置进行修改，同时其大量需求数据（地区，设备情况，区域内动物情况，设备实时状态，设备检测到的疑似目标）来自于后端

网页通过 Echarts 保证数据展示的实时性、流畅性以及美观程度，同时使用 DataV 快速开发，方便项目能够快速成型并二次迭代。

项目网页数据全部可以通过后端传输，保证数据的自由性，数据计算和验证也有完整处理。对于藏区的分区处理、多点记录和多设备情况整合已经完整实现。

此外，项目已经完成部署，目前已经部署有具体服务网站

（https://animal-managers-tibet.99pika.com）。

图 1.16 藏区珍稀动物检测系统

第二章 下一步工作计划

由于项目开发时间有限，本系统在开发过程中必然还存在着一些不足。因此有待之后进行更深入的改进和开发。我们下一步的工作计划如下：

① 与藏大等机构深入合作，完善系统数据源

目前本系统的数据源只来源于成都大熊猫繁育研究基地与西藏自治区高原生物研究所的动物真实数据集以及网上开源的珍稀动物无人机数据集，数据源相对较为局限。目前本系统在识别并跟踪野生动物的方面取得了较好的效果，但由于野生动物稀缺难寻且可用数据集太少，无法有效的评估本系统在此方面的可靠性和准确度，因此要加强与西藏大学等有关机构的合作，以获取更多当地珍稀动物的无人机视频数据，来完善系统数据源，使得本系统具有更

高的实际应用价值。

② **改进模型训练速度**

目前系统的相关模型训练速度不能很好的满足系统大规模应用于生态科研的需求，因此后期还可以继续优化算法，提升模型训练速度，在已有的识别和跟踪效果上实现进一步提升。

③ **实际效果测试**

在训练好识别和跟踪效果较好的模型后，我们打算将其实际放到野外环境中进行全方位的测试，并根据野外环境中可能出现的情况，如刮风、下雨等天气状况来改进模型。并根据实际测试的结果结合各个老师和专家的建议，对系统进行调整改进。

④ **项目成果转化**

在进一步完善系统功能之后，我们将继续对项目的方方面面进行相关的改善，优化项目的目前所得成果，积极参加相关的比赛并获得更多奖项。

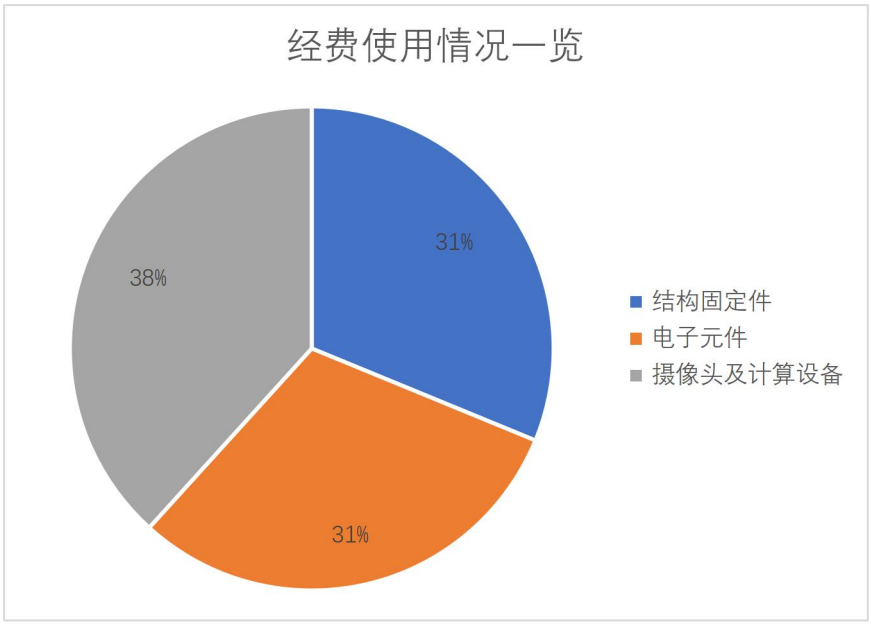
并重视系统在生态科研中的实际应用，实现全天候检测和追踪的功能，为野生动物保护贡献一份力量。

同时我们将对所有的项目文档进行整理，并将系统拿去**申请专利**。并围绕我们的识别和追踪模型，**发表相关的论文**。

第三章 经费使用情况和经费安排计划

3.1 经费使用情况

由于本系统要搭建一个可移植性强的云台摄像头系统，除了相关软件模型的开发，还有硬件系统的搭建。故经费大部分用于金属部件、微电子元件、摄像头等方面，用来搭建云台摄像头系统。至于软件部分，团队所在平台已提供了性能较强的CPU、服务器等，故无需再多购置。



具体开销如下：

经费使用情况一览					
结构固定件		电子元件		摄像头及计算设备	
物品	价格	物品	价格	物品	价格
弹簧	27.46	降压模块	165	摄像头	96.82
铝柱、套管、螺母	274.65	小电流模型插头	175	openmv摄像头	578
3d打印耗材	142	贴片电容	197	openmv延长线	117
文具	119	电子元器件	140	500N测力计	166
轴承	224.77	舵机舵盘	93.8	拉铆枪	8.5

图 大创经费使用情况一览

到目前为止，团队经费执行度为 99.8%。

3.2 经费安排计划

- **野外实地调研费用** 由于要进一步提高系统的鲁棒性，需要在实际的野外环境中实地调研，物资（如搭建好的可移植云台摄像系统等）运输需要一定的费用。
- **创建自己的数据集** 系统投入实际应用需要更多的野生动物数据集，除却校际机构的合作，还需要从国内外网络平台上下载相关的动物数据集，可能要支付一定的版权费。
- **参加比赛费用** 后期我们的系统仍然需要参加相关的比赛，可能需要报销参

加比赛的费用，如交通费、住宿费、申报费等。

- **申请专利或发表论文** 我们打算根据项目成果和项目理论方面的知识申请专利和发表论文，需要相应的申请费和版面费。

第四章 存在问题及需要说明的情况

4.1 存在问题

- **缺少珍稀动物数据集**

目前存在的最大问题，就是缺少珍稀动物的无人机数据集。从云台摄像头系统的物理结构到相关算法模型的搭建，我们已经开发出较成熟的目标识别与跟踪系统，并在识别并跟踪动物方面取得了较好的成果。虽然我们已有

来自成都大熊猫繁育研究基地与西藏自治区高原生物研究所的动物真实数据集，但珍稀动物数据集的数量不足，目前系统在动物识别跟踪方面的效果难以评估，模型效果不具普适性。我们要加强和西藏大学等有关机构的合作，获取更多的训练集，并据此调整模型参数，增加模型鲁棒性。

- **尚无无人机飞行许可证**

由于使用无人驾驶航空器从事航拍等活动，需要获得空域管理部门的批准，所以我们的系统需要申请到飞行许可证，才能进行真实野外环境的模拟测试。

4.2 需要说明的情况

为了更好的测试系统在野外的真实效果，团队决定暑期前往藏区进行实地考察测试，结合当地的情况对系统进行进一步的改进。