基于 PYNQ 的 AI 游戏系统

童博涵；李子辰

第一部分 设计概述

1.1 设计目的

使用 PYNQ 开发板实现游戏决策和操作系统，使 PYNQ 通过“看到”游戏图像做出决策并通过外设控制游戏操作，达到超越人类选手稳定性和成绩的水平。

1.2 应用领域

可供学习、为智能交互式设备提供创作思路、为使用人工智能或传统方法制作智能游戏机器人提供思路和样品

1.3 主要技术特点

1. 使用板载 HDMI IN 接口获取游戏图像数据，可直接连在电脑的 HDMI 输出接口上。
2. 使用板载 HDMI OUT 接口输出游戏过程中的调试数据和图窗。
3. 使用板载 PMOD 接口输出控制继电器驱动模块动作的信号，进一步通过继电器控制电磁铁动作敲击键盘，模拟人手操作。

1.4 关键性能指标

在实测中，使用 PYNQ Z2 开发板平台，对 Google Chrome Dino 小游戏进行测试。仅是第一版算法就跑出了 3000 分左右的游戏成绩，处理速度快、动作延时小。

1.5 主要创新点

1. 该设计中，开发板能“看到”的就是游戏的显示图像，而并非通过 API 等方式获取游戏参数。
2. 使用 3D 打印技术制作电磁铁阵列框架，模拟人手操作。
3. 开发板通过控制电磁铁，模拟人手的击键操作，而非通过 API 将操作传入游戏。
4. 可使用 PL 对运算过程实现硬件加速，提高处理速度。
5. 整套架构建立以后，可针对不同游戏进行优化算法，适配多种小游戏。

第二部分 系统组成及功能说明

2.1 整体介绍

本系统基于 PYNQ-Z2 开发板，实现了小游戏 Google Chrome Dino 的自动控制功能。使用 HDMI-IN 接口获取来自 PC 的输出视频流，通过 AXI 总线传入 PS 端中的图像处理模块并通过 HDMI-OUT 输出处理后图像，之后根

据处理结果通过 PMOD 控制继电器的开闭，进而驱动电磁铁起到敲击键盘控制游戏的效果。硬件框图如图 2.1 所示。

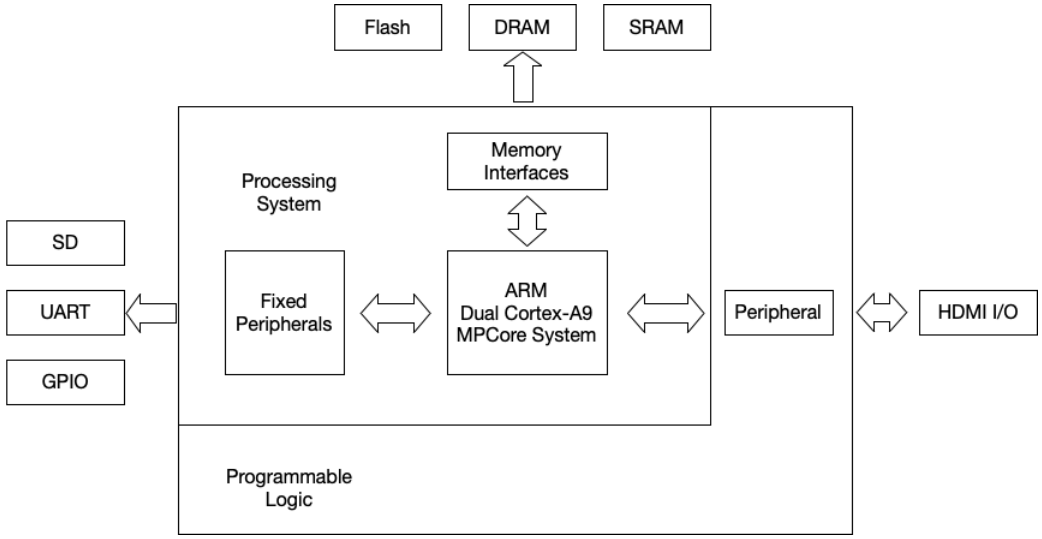

图 2.1 硬件框图

2.2 各模块介绍

(1) 图像处理模块

本模块在 PS 端基于 OpenCV 进行图像处理操作，从 HDMI-IN 中获取输入视频流，并将结果输出至 HDMI-OUT 和 PMOD 中。处理单帧的具体流程如图 2.2 所示：

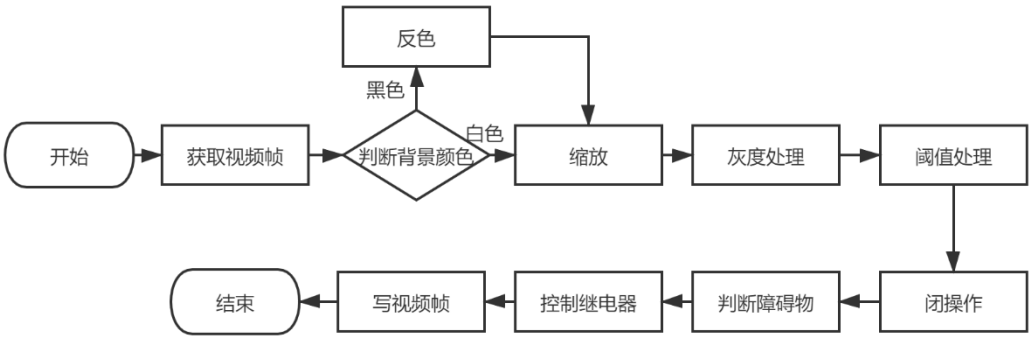

图 2.2 处理流程图

图像处理模块将获取到的视频帧先进行以下预处理：缩放和灰度操作将获取到的图像转换为灰度图像，以便于加快后续图像处理操作的速度；阈值操作将图像进行二值化操作，将灰度像素转化为 0 和 1，通过此操作可将输入图像中的游戏背景、地平线等干扰项消除；之后通过对图像作闭操作，可将二值化图像中的细小空洞进行填充，起到平滑轮廓线的作用。此外，经实验表明，在游戏运行一段时间之后，游戏背景将会由原来的白色变为黑色，因此在预操作之前加入判断游戏背景，若为黑色则将图像颜色反转。

预处理之后，在图像中控制角色的前方设置矩形边界，判断此时矩形边界中是否有像素点的值为 0，即出现障碍物，即可通过 PMOD 向继电

器输出控制信号。在所有处理操作完成之后，将游戏原始图像和处理后图像显示在 HDMI 显示屏上，并进行下一帧的处理操作。

(2) HDMI 输入输出模块

本模块通过 Overlay 调用 HDMI 中的输入和输出模块，并通过调用 pynq.lib.video 库实现了 HDMI 接口的控制，其中 HDMI 输入的参数设置为 1280*720 分辨率，模式为 24 位 BGR，输出参数为 1920*1080 分辨率，模式为 8 位灰度。

(3) 键盘控制模块

该模块由继电器驱动器、电磁铁、3D 打印支架组成。

继电器驱动器连接 PYNQ 开发板的的 PMOD 输出接口，由板载 Raspberry Pi 接口的 5V 针脚供电。当 PMOD 输出 3.3V 逻辑高电平时，继电器动作，将 24V 驱动电压送给电磁铁，电磁铁动作敲击键盘，完成一次键位触发。经过 100 毫秒延时后，PMOD 输出逻辑 0，继电器断开，电磁铁重新吸合。

继电器支架用于固定继电器相对于键盘的位置，以便准确点击按键。此框架是 3D 打印组件，其模型和实物图如图 2.3，图 2.4 所示，形状及制作均由本队自行完成。目前制作的框架可提供键盘方向键上下或空格键的操作。由于 3D 打印的便捷性，可快速重构并制作出不同的针对不同游戏的框架。

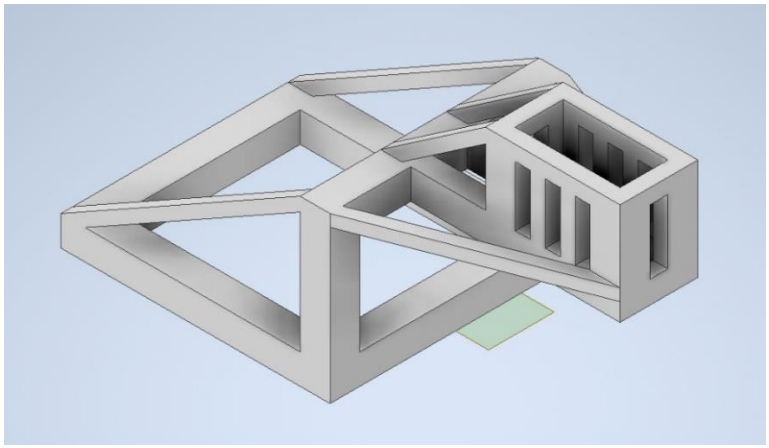

图 2.3 继电器支架 Inventor 模型

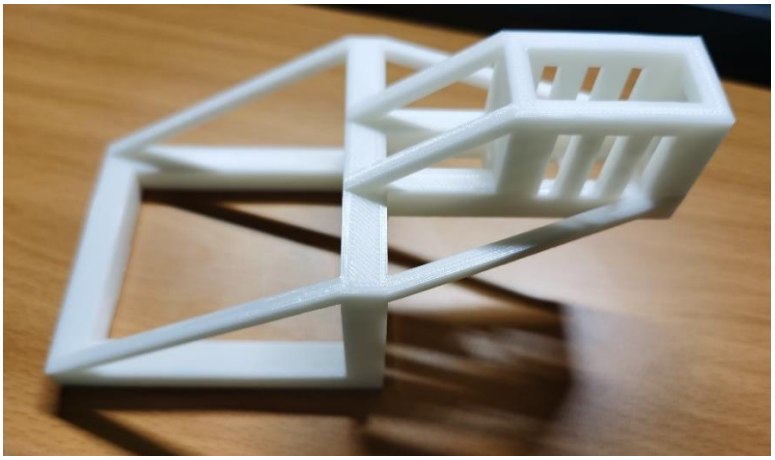

图 2.4 继电器支架实物图

第三部分　完成情况及性能参数

当前实现的系统实物图如图 3.1-3.4 所示，当前的游戏运行最高分为 3000 分左右，HDMI 输入输出、键盘控制模块正常运行。

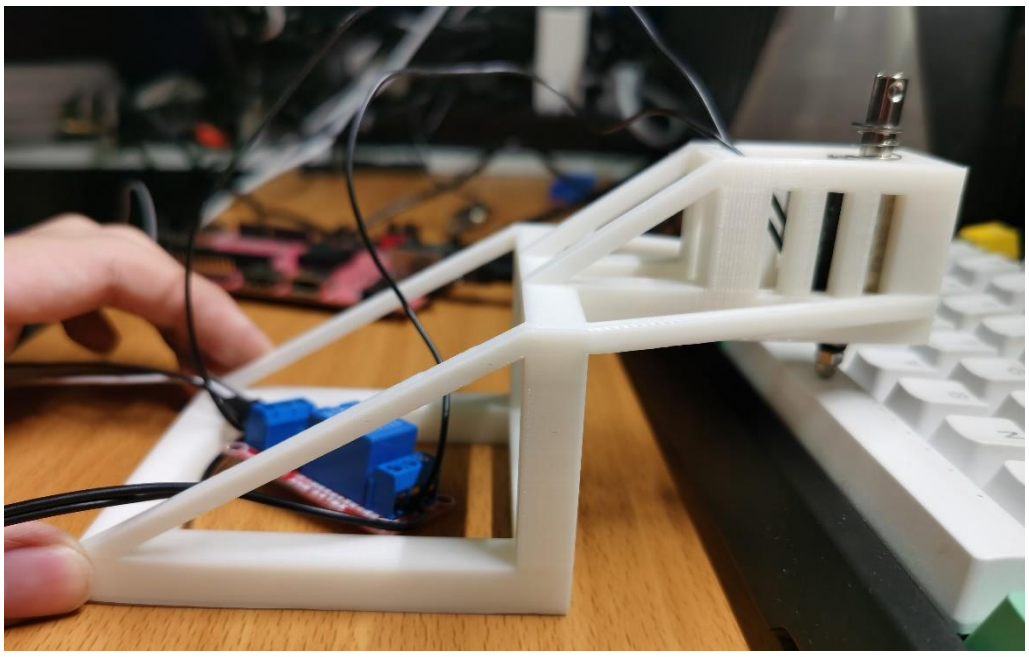

图 3.1 键盘敲击模块

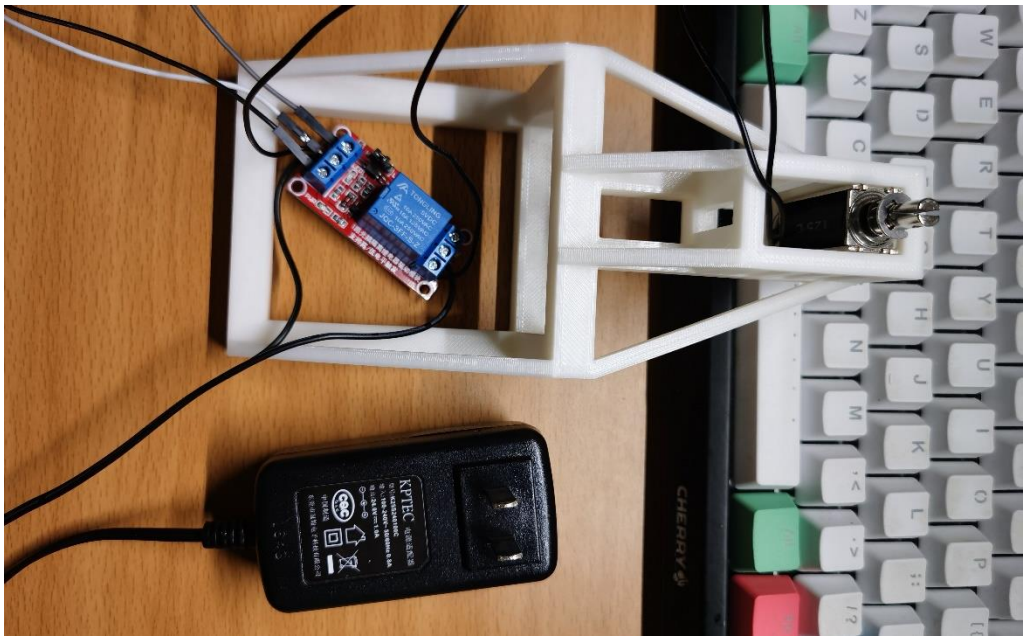

图 3.2 键盘敲击模块

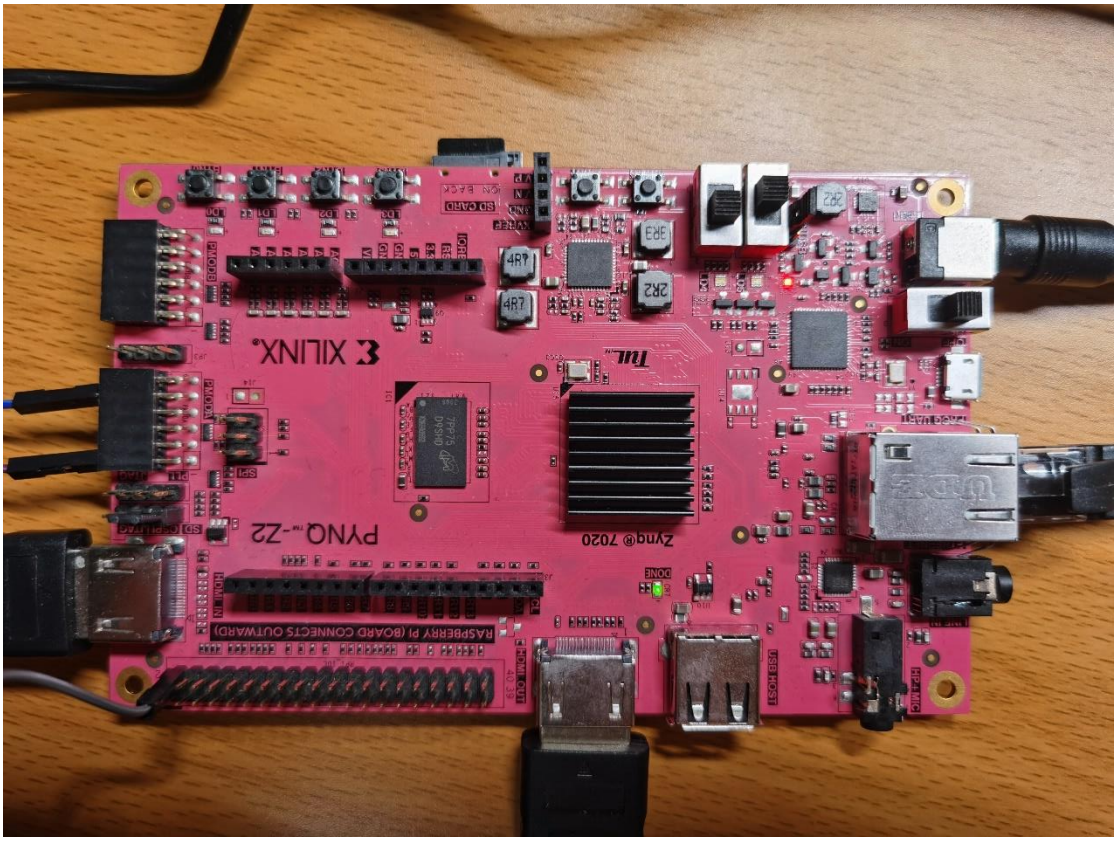

图 3.3 PYNQ-Z2 开发板

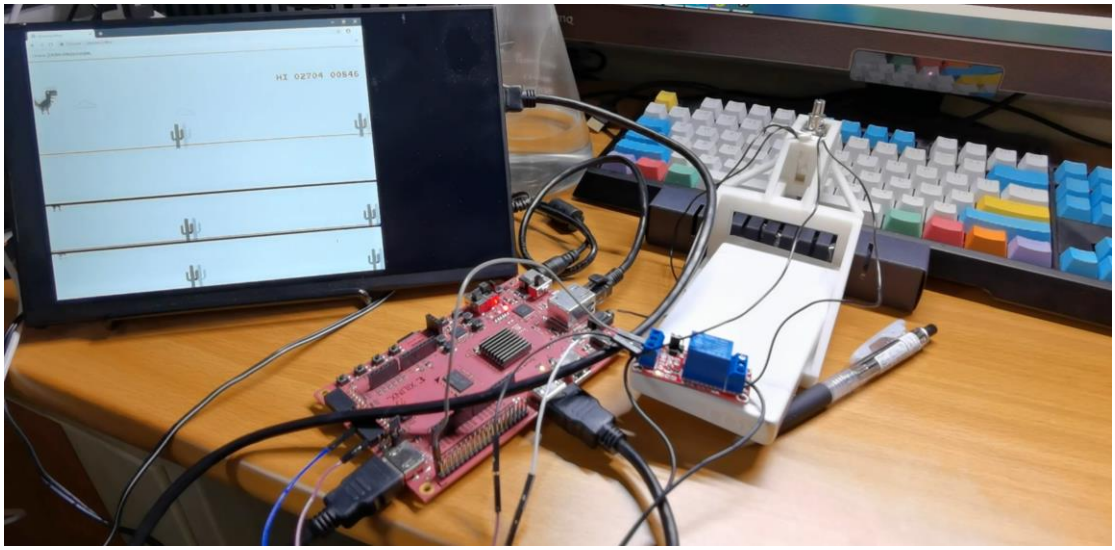

图 3.4 整体实物图

第四部分 总结

4.1 可扩展之处

本系统图像处理部分目前在 PS 端运行。在后期迁移至其他游戏时，可使用 Vivado HLS 工具和 Vitis Vision 库进行图像处理 IP 设计，并自定义 Overlay 实现图像流水线操作，实现算法的硬件加速。

对于除 Chrome-dino 之外逻辑更复杂的游戏，如 Edge Surf、超级马里奥等，需设计鲁棒性更高、处理速度更快的算法进行游戏控制。可使用深度强化学习方式代替传统图像处理方式，在 PL 端实现神经网络的硬件加速，PS 端负责控制 HDMI 输入和网络构建。

4.2 心得体会

做开发需要沉得住气，需要具备在压力下仍能处变不惊、冷静解决问题的能力。同时还必须具备超强的综合素养、学习能力和对新事物敏捷上手的能力——因为你永远无法知道当问题无法解决时，你的下一个思路会对你提出何种要求。

DDL 是第一生产力。

第五部分 参考文献

[1] 章毓晋. 图像工程（上册）：图像处理[M]. 北京：清华大学出版社，2012.
[2] https://pynq.readthedocs.io/en/latest/index.html
[3] https://docs.opencv.org/

第六部分 附录

主程序代码：

```
from pynq.overlays.base import BaseOverlay

from pynq.lib.video import *

from pynq.lib import Pmod_IO

import cv2

import time

import numpy as np


class DinoAgent:

    def __init__(self):

        base = BaseOverlay("base.bit")

        print("Base Initialized")

        self.pmod_up = Pmod_IO(base.PMODA, 0, 'out')

        print("pmod A Initialized")

        self.pmod_down = Pmod_IO(base.PMODB, 2, 'out')

        print("pmod B Initialized")
```

```
        frame_out_w = 1920
        frame_out_h = 1080
        #Mode_in = VideoMode(frame_out_w, frame_out_h,24)
        Mode_out = VideoMode(frame_out_w, frame_out_h, 8)
        self.hdmi_in = base.video.hdmi_in
        self.hdmi_in.configure()
        print(self.hdmi_in.mode)

        self.hdmi_in.start()
        print("HDMI In Initialized")

        self.hdmi_out = base.video.hdmi_out
        #self.hdmi_out.configure(Mode_out,PIXEL_RGB)
        self.hdmi_out.configure(Mode_out,PIXEL_GRAY)
        self.hdmi_out.start()
        #print(self.hdmi_in.mode)
        print(self.hdmi_out.mode)
        print("HDMI Out Initialized")
        #self.hdmi_in.tie(self.hdmi_out)
        self.flag=1

        print("Game start")
        self.jump()
        self.flag=0
        self.bitwise_mode = 0

    def jump(self):
        if self.flag == 0:
            self.pmod_up.write(1)
        else:
```

```
            self.pmod_up.write(0)


    def duck(self):
        self.pmod_down.write(0)
        #time.sleep(0.1)
        self.pmod_up.write(1)


    def video_pipeline(self):
        try:
            while True:
                try:
                    inframe = self.hdmi_in.readframe()
                    inframe_gray = cv2.cvtColor(inframe, cv2.COLOR_BGR2GRAY)
                    ret, thr = cv2.threshold(inframe_gray, 150, 255, cv2.THRESH_BIN
ARY)
                    thr = thr[300:480,:]
                    if ((thr[40:50,1100 ])==0).all():
                        cv2.bitwise_not(thr,thr)
                        self.bitwise_mode = 1
                    else:
                        self.bitwise_mode = 0
                    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
                    thr1 = cv2.morphologyEx(thr, cv2.MORPH_CLOSE, kernel)
                    self.judge(thr1)
                    #cv2.rectangle(thr,(240,0),(480,375),(255,255,255),2)
                    outframe = self.hdmi_out.newframe()
                    outframe[10:680,100:1380] = inframe_gray[:670,:]
                    outframe[690:870,100:1380] = thr


                    cv2.line(thr1, (1100, 40), (1100, 50), (0, 0, 0), 2)
```

```
                outframe[880:1060,100:1380] = thr1


                inframe.freebuffer()
                self.hdmi_out.writeframe(outframe)
            except KeyboardInterrupt:
                self.hdmi_out.close()
                self.hdmi_in.close()
                print('close pipeline')
                raise
            except ValueError:
                self.hdmi_out.close()
                self.hdmi_in.close()
                print('val err close pipeline')
                raise
        except KeyboardInterrupt:
            self.hdmi_out.close()
            self.hdmi_in.close()
            print('close pipeline')
            raise


    def judge(self, img):
        if self.bitwise_mode == 0:
            img_slice = img[135:180,200:240]
        else:
            img_slice = img[135:180,200:270]
        if((img_slice==0).any() and self.flag==0):
            self.jump()
            print('jump')
            self.flag=1
```

```
        elif((img_slice==0).any() and self.flag==1):
            self.jump()
            print('not jump')
        elif((img_slice==255).all()):
            self.flag=0



if __name__ == '__main__':
    dino = DinoAgent()
    dino.video_pipeline()
```