文章编号：2095-2163(2023)04-0060-09 中图分类号：TP242 文献标志码：A

基于软件定义的无人车系统

郑凯强
（浙江理工大学 信息学院，杭州 310018）

摘 要：无人车在实验和实际工作中发挥着重要作用。然而，在部分场景下，自主控制的无人车有时并不能有效按照用户的意愿执行任务，并且对于繁琐的软硬件以及接口管理是一件让专业人员都头疼的事情，同时有些危险多变的操作环境也对操作人员提出了挑战。所以也迫切需要一个操作简单、可以远程控制并且带有管理属性的无人车系统，即便是非专业用户依旧可以直观应用并快速入手。为此，在本文中，借鉴了软件定义无人系统的概念，对无人车的硬件进行抽象并构建控制接口以适应无人车底层硬件。设计一个能使用 VR 控制的无人车系统。同时在设计的应用场景中，对原有的导航过程中易出现的问题进行处理并达到了预期的解决效果。

关键词：软件定义；机器人操作系统；虚拟现实；数字孪生；动作捕捉技术

Unmanned vehicle system based on software definition

ZHENG Kaiqiang
（School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou 310018, China）

【Abstract】 Unmanned vehicle plays an important role in the experiment and practical work. However, in some scenarios, autonomous unmanned vehicles sometimes can not effectively carry out tasks according to the wishes of users, and cumbersome software, hardware and interface management are a headache for professionals. Apart from this, some dangerous and changeable operating environments also pose challenges to operators. Therefore, an unmanned vehicle system is urgently required that can be operated simply, remotely controlled and has management attributes. Even unprofessional users can still intuitively apply and start quickly. To address the problem, this paper uses the concept of software to define the unmanned system for reference, abstracts the hardware of the unmanned vehicle, and constructs the control interface to adapt to the underlying hardware of the unmanned vehicle. After that,an unmanned vehicle system is designed that can be controlled by VR. Meanwhile in the designed application scenario, the problems easy to appear in the original navigation process are handled, and the expected solution effect is achieved.

【Key words】 software definition; ROS; virtual reality; digital twins; motion capture technology

0 引 言

随着万物互联的时代的到来，大量的软硬件资源相互联通。各种设备在互联网上连接，各种新型的应用需求层出不穷，各类新的数字经济模式也陆续涌现，诸如电子商务、电子政务、共享经济、人工智能应用等。为了把各类软硬件的基础设施完成数字化以及定制化，就要用软件去重新定义传统的基础设施。目前，关于软件定义用得最多的就是软件定义网络(SDN)[1-2]，这是将网络分为 3 层，分别是：基础设施层、控制层和应用层。其中，控制层通过接口与基础设施层通信，向路由器和交换机发送数据包，同时向网络管理者接口提供网络相关的可编程接口，网络管理者是通过接口编写的管理程序来设定控制层，从而控制整个软件定义网络中的数据发送。

如今，软件定义无人系统的实际应用仍是一项难题。例如，在 DARPA 中的机器人技术挑战[3]或无人机控制[4]。无人系统有着各种软硬件设施以及不同式样的接口与编码方式，若想在动态变化的环境下调配和维护好这些资源并非易事 ，因此亟待使用软件定义范式抽象底层内容来控制底层设备，同时给用户带来便捷高效体验。本文的重点是提出一个可行的软件定义无人车系统来支持提供对资源的有效管理和控制或调整工作流以满足用户相关需求。

作者简介：郑凯强(1997-)，男，硕士研究生，主要研究方向：虚实交互系统、数字孪生。
通讯作者：郑凯强 Email：851576016@qq.com

收稿日期：2022-05-20

本文中，提出了软件定义无人车系统的主要架构，使用 MAPE-K 结构设计了软件定义无人车系统的控制层。其次，设计了搭建此系统的思路并据此思路开发搭建了一个有完整功能的无人车系统。然后，对本文系统进行导航和碰撞实验。最后，对本文所提系统进行了总结，并探讨了未来可能的改进方向。

1　软件定义架构

1.1　软件定义无人车的系统架构层次设计

本系统参照软件定义网络的结构，将系统框架分为 3 个层次：硬件执行层、控制器层和应用层[5]。其中，硬件执行层包含了无人车的各种硬件：激光雷达、里程计、路由器、移动底盘、车轮、非智能化电路板等，通过调用相关硬件可以实现用户需求。控制器层包括 3D 模拟系统、机器人操作系统（ROS）和动作捕捉系统。控制器层隐藏了与用户的需求无关联的硬件细节和硬件物理构造，如：激光雷达的控制信号、射电激光的口径、正常工作时的转速等，然后将硬件的基本信息、功能属性、使用属性等抽象出来并显示在应用层上，与此同时，控制器层提供了相关接口用来管控硬件执行层中基础硬件。应用层是管理用户交互的一层，应用层将感知并分析用户需求以维持系统与用户的交互，通常由一系列具体功能所组成。软件定义的无人车框架如图 1 所示。

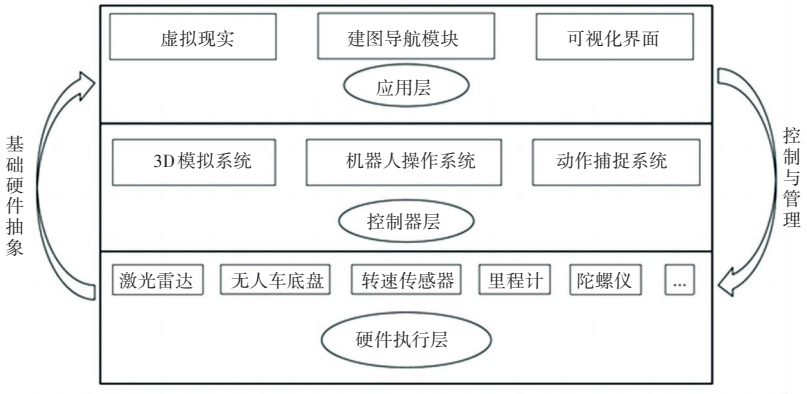

图 1　无人车架构

Fig. 1　The architecture of the unmanned vehicle

1.2　软件定义无人车系统核心设计

控制器层是整个无人系统的关键。控制器层提供接口完成对下层资源管理和控制。本文无人车的控制器层的设计上使用 MAPE-K 结构[6]，使管理器能够监视和控制所需硬件以及下位机。MAPE-K 结构如图 2 所示。由图 2 可知，MAPE-K 结构由 4 个组件和一个知识库组成。这 4 个组件是：监视组件、分析组件、计划组件和执行组件。

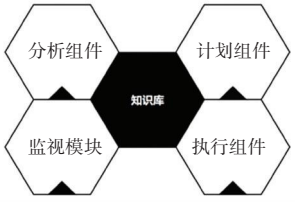

图 2　MAPE-K 结构示意图

Fig. 2　MAPE-K structure diagram

1.2.1　监视组件

监视组件（Moniter）在控制器层中起到监视作用，关于所监视的内容主要有 2 个，即：用户需求和系统运行时的环境。那么在软件定义的无人车系统里运行环境即无人车所处的环境，而用户需求即用户所期望无人车的行动。当然这些信息可能是会随时变化的，而这些改变是需要监视组件随时捕捉到的。为了对系统中的每一个变化都敏感，监视器可以由任何变化事件触发、且可随时随地触发。监视组件是系统变更的依据，系统可以根据捕获的变更收集需求变更（collectREInfo）、环境变更（collectENInfo）或资源变更（collectRSInfo）[7]。

在本系统中负责用户需求变更收集的是操作界面交互和 VR 设备交互。前者明确用户的导航需求，而后者则负责用户对无人车的移动需求。

以上是关于需求变更的内容。而由于环境变更对无人车的影响非常重要，所以关于环境变更，此软件定义无人车系统用了 ROS 和动作捕捉系统来监视随时发生的变化。在这其中 ROS 主要负责监视挂载在无人车上的传感器所获取的信息，例如激光雷达的扫描信息。而动作捕捉系统则负责环境信息矫正（矫正 ROS 上导航系统的偏差）和精确环境信息（主要是光学定位，会比 ROS 上的定位要更为

准确)。动作捕捉系统对无人车位置的捕捉如图 3 所示。

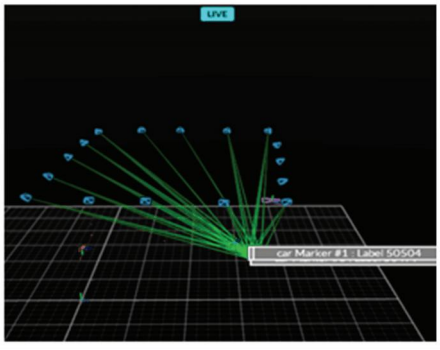

图 3 系统监测过程图

Fig. 3 System monitoring process diagram

1.2.2 分析组件

分析组件(Analyze)主要负责处理监视组件所收集到的信息，当系统或是环境出现一些变化时，由监视组件负责检测，分析组件则从监视组件收集的数据中提取有用的信息。为此，分析组件从需求和环境信息中提取重要特征，交由计划组件将进一步使用这些特征来做出决策。关于分析组件的运行过程，本文用 RVIZ 上的坐标转化来举例，相对坐标示意如图 4 所示。

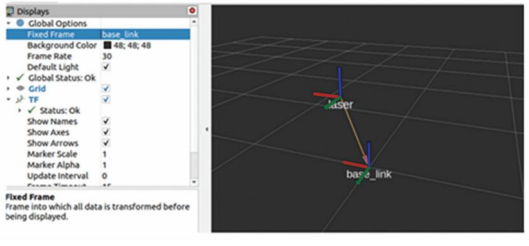

图 4 相对坐标示意图

Fig. 4 Schematic diagram of relative coordinates

雷达在底盘坐标系的位置和方向，可由如下公式进行描述：

$$[x_{cr} \quad y_{cr} \quad z_{cr}]^{\mathrm{T}} = \boldsymbol{R}_r[x_{wr} \quad y_{wr} \quad z_{wr}]^{\mathrm{T}} + \boldsymbol{T}_r \quad (1)$$

RVIZ 组建时雷达在世界坐标系的位置和方向，对此求得的世界坐标系为：

$$[x_{cl} \quad y_{cl} \quad z_{cl}]^{\mathrm{T}} = \boldsymbol{R}_l[x_{wl} \quad y_{wl} \quad z_{wl}]^{\mathrm{T}} + \boldsymbol{T}_l \quad (2)$$

由此推得：

$$[x_{wl} \quad y_{wl} \quad z_{wl}]^{\mathrm{T}} = \boldsymbol{R}_l^{-1}[x_{cl} \quad y_{cl} \quad z_{cl}]^{\mathrm{T}} + \boldsymbol{R}_l^{-1}\boldsymbol{T}_l \quad (3)$$

所以，在 RVIZ 使用 $\boldsymbol{R}$、$\boldsymbol{T}$ 矩阵时，雷达的旋转和平移公式见式(4)：

$$\begin{cases} \boldsymbol{rotation} = \boldsymbol{R}_l^{-1} \\ \boldsymbol{position} = -\boldsymbol{R}_l^{-1}\boldsymbol{T}_l \end{cases} \quad (4)$$

假设传感器能检测到雷达的相对坐标 $A(a, b, c)$，此坐标是以底座的坐标作为基准坐标系来定的。现有的无人车需要到达目标地，当前无人车的激光雷达可以探测到目标物的坐标 (x, y, z)，但是该坐标是以雷达为参考系的，而实际操作的是无人车的底盘。此时需要调用 ROS 中的 TF(Transform Listener)节点来负责分析雷达到底盘的坐标转化关系。

1.2.3 决策组件

决策组件(Plan)会根据分析组件提供环境信息和系统资源的状态做出决策。决策的目的就是使所有的决策都在系统所能提供方式里尽量满足用户的需求。寻求一个在当前状况下的最优解，在当前系统配置仍然能满足用户需求的情况下选择不变更系统的决策，反之，则更改系统的决策。例如：当无人车在没有靠近障碍物时系统使用的是全局路径规划 global_planner 节点，但当无人车很接近障碍物时调用的将会是局部路径规划 local_planner 节点，碰到障碍后调用的是自动进行逃离恢复 recovery_behaviors 结点等。

1.2.4 执行组件

执行组件(Execute)根据上一步分析组件的安排对系统中的资源进行调度。执行组件会将需要调用的组件写成一个工作流程。通过这个流程来实现资源的调度。

ROS 对无人车的下位机传输运动控制指令前，需要获取 odom 话题中里程计信息以及 scan 话题中的激光雷达信息，然后通过 cmd_vel 话题中的下位机控制接口来实现控制指令发送，ROS 硬件控制框架如图 5 所示。

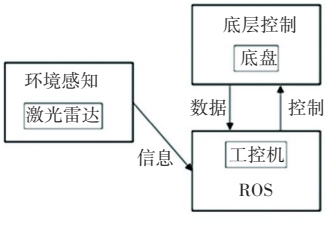

图 5 ROS 硬件控制框架

Fig. 5 ROS hardware control framework

cmd_vel 话题将控制消息传输到下位机控制接口从而对无人车底盘进行运动控制，该消息的主要类型为 geometry_msgs/Twist，该数据类型最常用的 3 个变量为：*linear.x*、*linear.y* 和 *angular.z*，分别表示水平线速度、垂直线速度以及角速度。

ROS 对无人车的下位机控制原理如图 6 所示，

有控制需求的节点通过 cmd_vel 话题向相关接口 base_controller 节点发送话题信息，然后 base_controller 节点会将控制信息通过串口传输给控制板（本文无人车中使用的是 Adurnio 控制板）。接下来，下位机根据相关运动过程的计算来确定无人车底盘轮胎的转速，并将转速信息发给电动机驱动以达到控制电机转速的效果。最后，电动机驱动会计算来自编码器的电子脉冲速度，以此对轮子转速进行监测并将转速控制在一个定值。

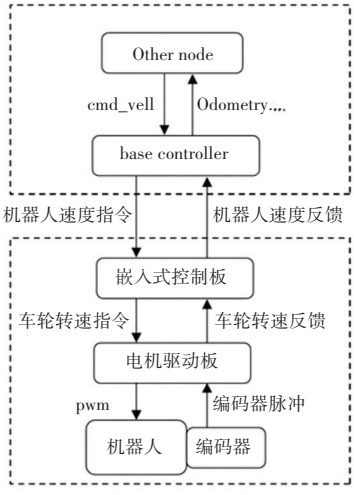

图 6　下位机控制原理图

Fig. 6　Schematic diagram of lower computer control

1.2.5　知识库

知识库类似数据库，主要负责维护服务于此系统的 4 种信息：可用软件服务列表、硬件设备列表、环境信息和系统配置信息。对于前 2 种类型的信息，其中环境信息和系统配置信息则属于一种动态信息。当一些现有的软件服务或硬件设备发生变化时，知识库会对这些变化进行处理和反映，使知识库中存储的信息始终与系统资源的状态保持一致[8-9]。

系统有时候需要接入一些新的设备或者软件服务，此时这些信息需要被注册在系统之中。注册软件服务或硬件设备时，需要将相关资源加载到系统中，并由软件资源池或设备层来维护。此时需要借助一个叫 RVIZ 的可视化工具，向用户展示传感器的数据信息、无人车模型、坐标变化关系等。要将相关的功能信息通过发送节点发送并部署到 ROS_displays 节点中，才能将硬件信息展示在 RVIZ 界面上。RVIZ 上激光雷达数据如图 7 所示。RVIZ 订阅了激光雷达数据消息，并展示激光雷达数据。

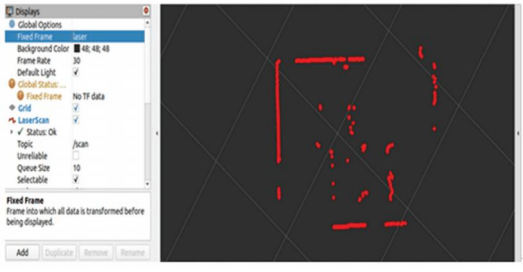

图 7　RVIZ 上激光雷达数据

Fig. 7　Lidar data on RVIZ

2　设计思路与开发流程

2.1　系统总体框架

系统采用 3 层架构体系，由显示模块、操作模块、功能模块、硬件模块、物理以及数字模型模块、监视模块等 6 个模块组成，如图 8 所示。这里，对系统中各模块的设计功能，将给出剖析概述如下。

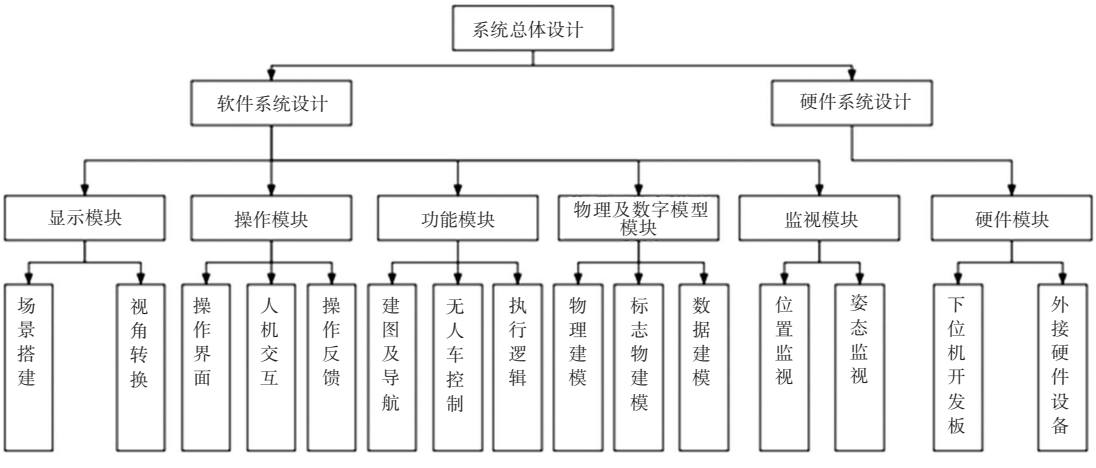

图 8　系统总体框架

Fig. 8　The framework of the system

(1)显示模块。用户可以在 VR 中看到的效果,包括场景搭建和视角转换。

(2)操作模块。该模块主要实现系统的 GUI 界面显示和人机交互等功能。

(3)功能模块。软件定义的核心模块,主要负责建图、导航、控制无人车移动以及执行逻辑等模块。

(4)硬件模块。小车的外接设备,以及下位机开发板。

(5)物理以及数字模型模块。在 Unity 中通过编写 C#脚本、动作捕捉系统中 3D 标志物,对小车的物理模型(Rigidbody)和物理外型的建模;在 ROS 中则是对小车外型和传感器等硬件的建模,以实现无人车的虚拟仿真。

(6)监视模块。动作捕捉系统对无人车和障碍物位姿的监视。

2.2 开发流程

软件定义无人车系统主要由 VR 系统、3D 模拟系统(Unity 引擎)、机器人操作系统(ROS)、动作捕捉系统、实体无人车五部分组成。该系统的开发流程如图 9 所示。对此拟做阐释分述如下。

Step 1 完成对实体无人车平台以及传感器的搭建。

Step 2 在 3DXMax 中对障碍物和虚拟无人车进行三维建模并赋予对应物理属性。

Step 3 在 Unity 中编写与实体无人车下位机通讯 SDK 以及运行逻辑,并与虚拟无人车部件对应起来。

Step 4 用 Unity 对无人车使用场景进行设计与开发。

Step 5 搭建 ROS,设置 ROS 中各结点通讯。

Step 6 用 ROS 对无人车下位机进行通讯,用 URDF 集成 RVIZ 建立可用模型、关节件之间的逻辑(用于计算导航的相对偏差)。

Step 7 在 ROS 中布置好 SLAM 建图和导航算法。

Step 8 在动作捕捉系统中标注跟踪点信息,配置跟踪数据,并用跟踪点建模刚体作为实体无人车在虚拟中所反映位姿的虚拟模型。

Step 9 让 Unity 与动作捕捉系统进行通讯,将动作捕捉系统中的刚体与到 Unity 的虚拟无人车模型进行实时通讯。

Step 10 配置好 VR 设备,并编写调试关于 VR 设备的交互系统,用 SteamVR 连接 Unity。

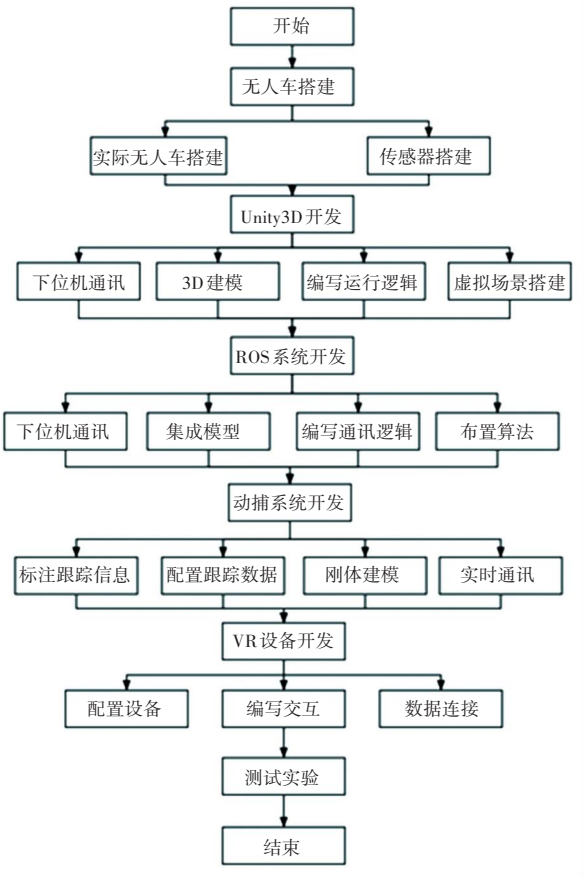

图 9 RVIZ 上激光雷达数据系统开发流程图

Fig. 9 Flow chart of the system development

3 系统模块介绍

3.1 虚拟现实

虚拟现实(VR)是以计算机技术为核心,集逼真的视、听、触觉为一体化的虚拟环境模拟系统。用户借助 VR 设备(如特制的头盔、手套、手柄等),以自然方式与虚拟环境中的客体进行交互,从而产生身临其境的体验[10],文中 VR 设备的组成有:1 个头戴式显示器和 2 个控制器。控制器是完全无线的,头盔显示器通过 USB 和 HDMI 线连接到电脑。每个控制器都有一个触摸板、触发器和 2 个用户输入按钮。VR 设备支持多个游戏和物理引擎,通过一个名为 SteamVR 的 SDK 连接到 Unity,且 VR 上显示的内容是在 Unity 上开发的。虚拟现实界面和机架系统也提供了直观的方法用于将用户的动作映射到正在控制的机器人[11]。VR 将模拟世界与物理世界相结合,在机器人仿真领域做出了贡献[12]。

3.2 机器人操作系统

机器人操作系统(ROS)是一种用于机器人的次级操作系统,包含底层驱动程序管理、硬件抽象描

述、共用功能的执行、程序发行包管理等功能，其首要设计目标是在机器人研发领域提高代码复用率。此外，ROS 利用简单、语言无关的接口定义语言去描述模块之间的消息传递，这样就可支持C/C++、Python、Java 等多种编程语言以实现跨平台、跨语言的交互。ROS 将执行不同功能的程序进程(称为节点)连接起来。节点通过在本地 TCP 网络(称为 ROS 网络)上的通道或主题流数据进行通信。节点创建 publisher 对象来在网络上发布话题上的数据，或者创建 subscriber 对象来订阅话题。

3.3 Unity

Unity 是一款可视化开发 3D 应用工具，采用 C#编程，易于开发，具有强大的开发者社区支持和第三方开发插件，兼容所有平台[13]，Unity 可以用于 3D 环境并允许传感器建模[14]。由于内置物理引擎，Unity 系统对接触动力学以及材料模拟均有相关技术支持。此外，Unity 引擎提供多个着色器和图形效果来增强 3D 虚拟现实的真实性和真实性环境[15]。

一个开放的 Unity 环境被称为场景。GameObject 是场景中的基本单位，本文所提出的虚拟无人车模型就是由许多个 GameObject 组成。附在每个 GameObject 上的是组件，有数十种类型的组件，组件上挂载各种脚本。脚本是一个小的 C# 程序，在每个渲染帧中执行。Unity 作为本文所提系统的模型和通讯核心的功能就是通过这些 C#脚本实现的。

3.4 无人车

本次实验使用的无人车是 DashGo D1，在此移动平台上的硬件有工控机、DashGo D1 移动底盘、Slam 导航模块、PS1000C 内置的 IMU 模块、高精度 G4 激光雷达和 600 线增量型光电编码器等。底盘采用先进的动力平台悬挂装置，有着优异的地面适应性能。无人车本体如图 10 所示。

图 10　无人车图

Fig. 10　Unmanned vehicle

4　系统搭建与实验过程

4.1　系统搭建及实验布置

软件定义无人车系统主要由 VR 系统、Unity 引擎、ROS、动作捕捉系统、实体无人车、五部分分组成。其中，VR 系统主要搭建 VR 视角以及用户的互动操作。Unity 引擎主要起到连接各系统、以及形成虚拟无人车和无人车的运行环境。ROS 用来完成无人车数据可视化和提供软件定义接口完成建图导航功能调用。

首先，3Dmarks 对无人车进行建模，进行贴图与渲染形成虚拟无人车模型。然后，用连接 VR 接收器，布置场地基站，配置好 Unity 上的 steamVR(作为 SDK)，连接 VR 设备(头盔和手柄)到 Unity 上。再给小车添加材质，添加模拟地形，添加摄像机，添加碰撞体，添加碰撞事件与监控。通过编写 C#脚本设置对实体无人车的、如软件定义控制接口。添加物理性质的刚体(Rigidbody)和建立物理模型，实现无人车运行的虚拟仿真。仿真结果如图 11 所示。

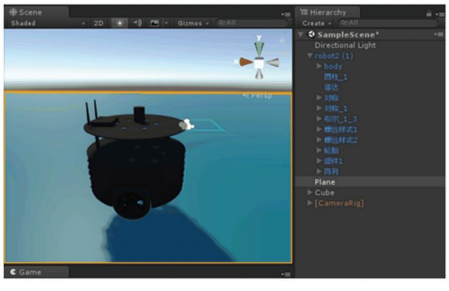

图 11　无人车主体以及附属的硬件建模

Fig. 11　Hardware modeling of unmanned vehicle

在 Unity 中用 OptiTrack_Unity_Plugin 通讯协议与动作捕捉系统进行通讯，接受动作捕捉系统传来的流捕捉数据与刚体模型，并将数据与虚拟无人车进行连接，以保证虚拟无人车能反映实体无人车和障碍物的位姿。与 VR 进行连接，编译用户交互逻辑和 VR 环境，保证虚拟图像能显示在 VR 头盔上。编写 python 脚本(附带 Arduino 用语的功能包)与实体无人车通讯，借助 Arduino 开发板获取小车控制信息。

关于动作捕捉系统，本研究给无人车与障碍物贴上标志物，对环境多次校准并用 XML 脚本来描述标志点位置与相对位置信息[9]。将标志点信息合成刚体 Rigid Body 跟踪模型，如图 12 所示。然后，将跟踪数据保存到文件中(TAK 文件)，捕获标

志点时，所有 2D 数据、实时重建的 3D 数据和求解的数据都会保存到文件中。最后，为传输数据流选择网络适配器(接口、IP 地址)，与 Unity 建立连接，并将 TAK 文件赋予虚拟无人车模型。

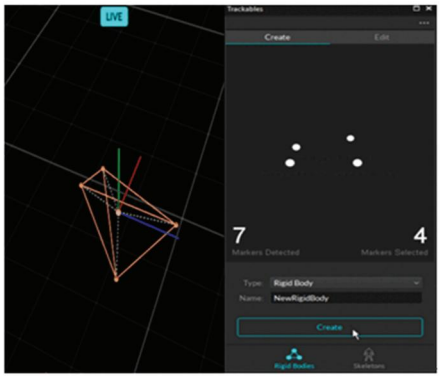

图 12 动作捕捉系统中的刚体模型

Fig. 12 Rigid body model in motion capture system

关于 ROS，在无人车的开发板上部署 ROS，并导入一些常用模块(收发通信、建模、坐标变换、操作界面等)，编译无人车各部件坐标变换逻辑代码，用 XML 编写并集成抽象无人车模型(URDF)各部分以及连接关系。将抽象过的各硬件数据情况部署到 RVIZ 上。在 ROS 上布置所需要用到的建图和自主导航算法。

4.2 导航实验

本实验的重点应用场景在办公室，主要由无人车和一些障碍物组成。然后，用户需要在地图上给定一个目的地和一个出发点(一般是无人车当前所在位置)。接着无人车按照既定的算法到达目的地，中途无人车可能会碰撞上障碍物，如果无人车碰撞上障碍物，用户就可以使用 VR 去控制无人车脱离障碍物。

在本文的实验中，使用了自己配置的无人车。用户使用 VR 设备于控制小车或者使用点击设置目的地的方式与导航系统进行交互。VR 可以对无人车的运动状况进行实时数字孪生虚模拟[8]，再将模拟好的环境和无人车模型显示在 VR 头盔上。在无人车运行的过程中也可以通过观测 ROS 中的 RVIZ 面板来了解小车的实时的状态，而不用通过专业的仪器运算以及实时监测。

图 13 为导航任务开始时的导航图，无人车导航系统已经部署了基于 A* 算法的全局规划和基于 DWA 算法的局部规划。从中可以看出导航系统对从起点到终点做出的全局规划线路，并使用局部规划对第一个地形进行绕行。图 14 为导航一段时间之后的观测图，可以看出局部路径规划已经在起作用，无人车已经顺利通过前段地形，并且图 14 中无人车尾部的 AMCL 粒子比较集中，说明此时位姿信息比较准确且定位系统运行稳定。

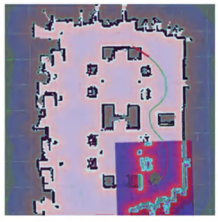

图 13 导航任务开始

Fig. 13 Navigation task start

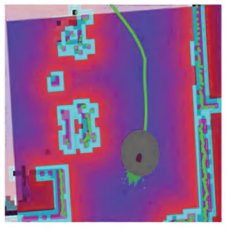

图 14 导航进行中

Fig. 14 Navigation in progress

在无人移动到图 15 显示的位置之前，为了测试导航中的局部路径算法是否能检测到障碍物位置变化，将箱子的位置向左移动。图 13 显示在 SLAM 所建的地图中，原先箱子摆放的位置靠墙。因为箱子左侧位置在全局规划时默认是可以通过的，所以按照全局路径规划默认从箱子的左侧位置绕行。图 15 显示无人车在箱子附近并未按照原有路线移动，因为无人车在 DWA 局部路径规划时检测到左前方有障碍物且原先的障碍物不在全局路径规划前的位置。可以通过观察图 15 浅蓝色框了解移动过的障碍物的大体位置，但是导航系统对障碍物的反映非常不明显，无法判断无人车在越过障碍物时的具体情况。随后 DWA 局部路径导航在综合分析环境和无人车位姿的情况下，引导无人车向右边绕过障碍物。图 16 显示无人车按照新规划的路线绕过了障碍物。

图 15 导航过程 1

Fig. 15 Navigation process 1

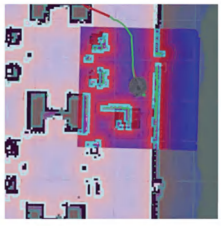

图 16 导航过程 2

Fig. 16 Navigation process 2

对比导航系统画面，参见图 17。由图 17 可看到，在 VR 视角下，由于动作捕捉系统对箱子位姿的捕捉，可以直观地看到箱子已经移动。继续观察 VR 画面，参见图 18。由图 18 可以观察到，局部路径规划的过程中无人车更改相关路线并向箱子右侧避障，在观察到无人车成功避障之后，因为无人车并未出现异常情况，故不需要用 VR 手柄做出控制操作。

图 17　VR 图 1

Fig. 17　VR figure 1

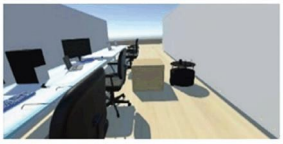

图 18　VR 图 2

Fig. 18　VR figure 2

在通过 DWA 局部路径规划成功绕过箱子障碍物之后，仿真结果如图 19 所示。在图 19 中，可以从 VR 中观察到无人车正在逐渐靠近终点。通过导航系统画面和本文系统框架中 VR 画面的对比，可以发现本文系统框架针对障碍物位置更改的即时反映效果和对无人车避障过程的直观性要优于仅使用导航系统画面的无人车系统。图 20 为导航过程实物图。

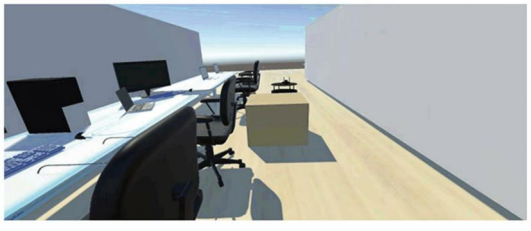

图 19　接近终点

Fig. 19　Near the end

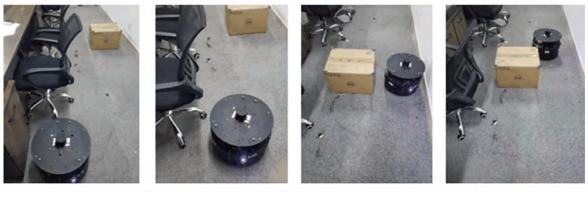

图 20　导航过程实物图

Fig. 20　Physical map of navigation process

4.3　碰撞处理实验

无人车碰撞问题是一种在无人车运行过程中的常见问题。当无人车没有出现在操作人员视野中，发生碰撞概率较大，即使在无人车上安装摄像头，由于摄像头视角的问题，解决碰撞问题难度也较大。本文系统通过动作捕捉系统对无人车和障碍物位姿信息的捕捉，能使用户更好了解碰撞情况和解决碰撞问题。

在实际场景中判断物体之间是否碰撞十分重要，所以在虚拟场景中模拟准确的碰撞过程不仅仅需要依靠动作捕捉系统的高速摄像头所获取的数据，还需要在 Unity 中添加碰撞检测。在 Unity 的虚拟无人车里加入的碰撞组件和物理刚体，能对无人车和障碍物碰撞时的物理过程进行模拟。结合动作捕捉系统的标志点信息组成的刚体位姿，能让 VR 中得出与真实碰撞情况一致的碰撞结果，以便用户在 VR 里就能观测到实际无人车发生碰撞后的具体情况，然后利用 VR 控制无人车离开碰撞区域。这样的设计改善了无人车碰撞后直接停滞和因使用者不了解具体碰撞情况而无法进行后续操作的问题。

当无人车与障碍物发生碰撞时，碰撞效果如图 21 所示。由图 21 可看到，导航界面对碰撞后的效果显示并不明显。通过获取动作捕捉系统的位姿观测信息，VR 能很好地反映实际的碰撞情况，下面以无人车碰撞箱子情况为例，来观察 VR 视角里的碰撞与实际碰撞效果：从图 21～图 23 的对比可以看到相较于普通导航显示界面，软件定义无人车系统的 VR 画面能更直观地向用户展示碰撞情况，且可发现碰撞情况的表现更加准确。在此基础上又研究得到，图 24、图 25 为动作捕捉系统中的无人车和障碍物的位姿变化。动作捕捉系统中位姿的变化验证了 VR 中无人车与障碍物相对位置关系的准确性。进一步得到，图 26、图 27 为实际碰撞情况。

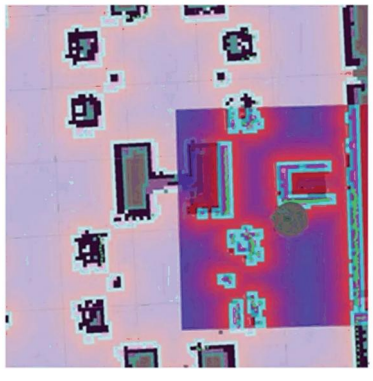

图 21　碰撞效果图 1

Fig. 21　Impact effect drawing 1

图 22　碰撞效果图 2

Fig. 21　Impact effect drawing 2

图 23　碰撞效果图 3

Fig. 23　Impact effect drawing 3

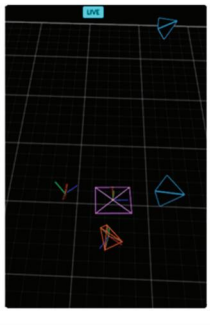

图 24　动作捕捉图 1

Fig. 24　Motion capture figure 1

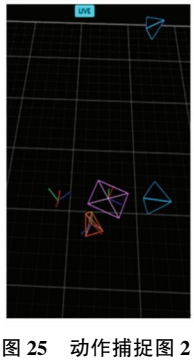

图 25　动作捕捉图 2

Fig. 25　Motion capture figure 2

图 26 实际碰撞情况 1

Fig. 26 Actual collision 1

图 27 实际碰撞情况 2

Fig. 27 Actual collision 2

无人车移动出碰撞区域如图 28 所示。由图 28 可看到，在发生碰撞后，由于在 VR 中能观察到具体碰撞情况，所以可通过 VR 手柄控制的方式，将无人车安全移动出发生碰撞的区域，从而避免后续问题。

图 28 无人车移动出碰撞区域

Fig. 28 Unmanned vehicle moves out of the collision area

5 结束语

为了验证该系统能否满足用户的简单要求，并测试系统对非专业用户的友好性。研究时邀请了非相关技术的 10 名学生对基于软件定义的无人车系统进行测试。通过与 10 名学生的共同测试与交流表明，该系统功能模块完整，并且无人车能够按照用户的远程的控制指令实时运动，无停顿感，系统运行流畅，具有较强的交互性能，导航功能顺畅。测试结果通过对软件定义的核心 MAPE-K 框架来评价系统的完成度。系统测试结果见表 1。

表 1 系统测试结果

Tab. 1 Test results of the system

测试指标	*M*	*A*	*P*	*E*	*K*
良好	8	9	9	7	8
一般	2	1	1	3	2
较差	0	0	0	0	0

本文设计了一个软件定义的框架。该框架涉及 2 个关键方面。一个是软件定义的软件服务和硬件设备，另一个是基于 MAPE-K 的控制器。前者用于对无人系统的软硬件资源进行抽象并提供控制接口，后者负责对资源进行管理，并对完成给定任务的工作流程进行调度。本系统对原有的自主导航系统来说，添加了监视器也就是动作捕捉系统，针对性地提高了监视器的精准度和导航过程的可验证性，此外还通过 VR 的方式增强系统对用户的交互感和使用体验，降低了新用户的上手难度。此系统也有一定的不足，首先是小车模型过于单一的问题，导致此系统软件定义资源虚拟化的时候并没有许多可供操作的内容，系统没有带有 3D 全景摄像头和图像识别的内容，用户交互界面比较简单。研究按照提出的软件定义体系结构为原型设计了一个无人车以及系统，并通过一个应用场景验证了其可行性。在将来，将通过考虑其有效性，以及无人系统的可靠性和可扩展性来做深入研究，则已成为下一阶段的工作。

参考文献

[1] 张朝昆，崔勇，唐嘉嘉，吴建平. 软件定义网络(SDN)研究进展[J]. 软件学报，2015，26(01)：62-81.

[2] BERA S，MISRA S，VASILAKOS A V. Software - defined networking for Internet of Things：A survey[J]. IEEE Internet of Things Journal，2017，4(6)：1994-2008.

[3] DELLIN C M，STRABALA K，HAYNES G，et al. Guided manipulation planning at the DARPA robotics challenge trials[J]. Advanced Robotics，2016，109：149-163.

[4] GNATZIG S，CHUCHOLOWSKI F，TANG T，et al. A system design for teleoperated road vehicles[J]. ICINCO ，2013(2)：231-238.

[5] KE Xu，WANG Xiaoliang，WEI Wei，et al. Toward software defined smart home[J]. IEEE Communications Magazine，2016，54(5)：116-122.

[6] OLIVEIRA R，SCHWEITZER C M，SHINODA A A ，et al. Using Mininet for emulation and prototyping software - defined networks [C]// 2014 IEEE Colombian Conference on Communications and Computing (COLCOM). Bogota，Colombia：IEEE，2014：1-6.

[7] 孟韶南，梁雁冰，师恒. 基于 ROS 平台的六自由度机械臂运动规划[J]. 上海交通大学学报，2016(S1)：94-97.

[8] 李其锐.浅谈数字孪生工厂的工艺流程三维模拟技术[J]. 数字技术与应用，2020，38(03)：73-74.

[9] 向泽锐，支锦亦，徐伯初，李娟.运动捕捉技术及其应用研究综述[J]. 计算机应研究，2013，30(08)：2241-2245.

[10] 赵沁平. 虚拟现实综述[J]. 中国科学：信息科学，2009，39(01)：2-46.

[11] WHITNEY D，ROSEN E，PHILLIPS E，et al. Comparing robot grasping teleoperation acROSs desktop and virtual reality with ROS reality [C]//International Symposium on Robotics Research. Puerto Varas，Chile：Springer，2017：1-16.

[12] LIU Yuzhou，NOVOTNY G，SMIRNOV N，et al. Mobile delivery robots：Mixed reality - based simulation relying on ROS and Unity 3D[C]//2020 IEEE Intelligent Vehicles Symposium (IV).Las Vegas，NV，USA：IEEE，2020：15-20.

[13] KYAW A S，PETERS C，SWE T N. Unity 4. x game AI programming[M]. Birmingham：Packt Publishing Ltd.，2013.

[14] 郭晓敏，申闫春. 基于 Unity/Vuforia 的 AR 导览系统研究[J]. 计算机仿真，2019，36(08)：165-169.

[15] CHRISTODOULOU S，MICHAEL D，GREGORIADES A，et al. Design of a 3d interactive simulator for driver behavior analysis [C]//Proceedings of the 2013 Summer Computer Simulation Conference. Toronto，Ontario，Canada：ACM，2013：17.