ROS-Based Control of the DJI Matrice 100 Robot with QR Images Obtained from DJI Guidance

Abdülkadir Çakır[#1], Seyit Akpancar[*2]

[#*]*Department of Electrical and Electronics Engineering, Faculty of Technology, Isparta University of Applied Sciences, Isparta, Turkey*

Abstract *This article discusses the development of a control software for the DJI Matrice 100 robot, which is attracting the interest and is seeing increased use among researchers studying flying robots. The instant QR images used in this study are obtained from the right camera attached to the VBUS3 port of the DJI Guidance system. In addition to the control software used for the processing of the QR images obtained from DJI Guidance, the study also made use of the ZBar Bar Code Reader.*

The development of a software for the operation of a DJI Matrice 100 robot facilitated the running or control of the software solutions created for the PC-controlled Matrice 100 robot, and removed the need for a PC screen operating under grid power during the test phases of the field studies.

This study helped in the development of an auxiliary software for researchers studying the DJI Matrice 100 robot. It is thought that the developed software will be used by the researchers in their future studies and it will become a highly addressed study in the literature.

Keywords — *ROS, DJI Matrice 100, QR Code, Controller.*

I. INTRODUCTION

Studies into robot technologies are becoming increasingly common, with a wealth of innovative solutions being developed in a broad range of fields, including military, health, search and rescue, and research. There has also been a significant increase in studies on the active use of air robots, which have no limitations on movement in areas that may pose a threat to human health or that may provide discomfort due to bumps, holes or inclines [1], [2], [3], [4].

During field studies, it is necessary for researchers to carry a PC monitor if they are make use of software solutions developed for the control of robots via PC. While the PCs used for the control of robots use the integral batteries of the robot platforms as a power source, PC monitors must use grid power, although this can lead to both higher costs and additional time for researchers.

II. FLYING ROBOT PLATFORM DJI MATRICE 100

Fig. 1 shows the DJI Matrice 100 robot that was used in the present study. The Matrice 100 is one of the most popular robot platforms among researchers of robotics due to of its high carrying capacity (around 1250 gr, other than its own weight), a choice of operating systems (Linux, Windows), conformity with ROS and the existence of a wide variety of SDK [5], [6], [7].

Fig. 1 Flying Robot Platform DJI Matrice 100

This study proposes a software that facilitates the control of the flying Matrice 100 platform by means of ROS.

III. ROS (ROBOT OPERATING SYSTEM) AND GAZEBO

A. Why ROS?

ROS was adopted in the present study for a number of reasons, including its common use in robotic projects and applications in the world of robotics, its recent introduction in literature, the flexibility of its development setting, its ability to be integrated into other systems that are developed in real settings, the existence of fewer experts on simulation and its free license.

ROS (Robot Operating System) was first developed by the Artificial Intelligence Laboratory of Stanford University as part of the STAIR (Stanford AI Robot) project in 2007 [8]. ROS functions as an operating system or a software framework that provides all the libraries and tools required by robot software developers, like a standard operating system such as HAL (Hardware Abstraction Layer). It allows for a lower degree of device control, data exchange

and package management [9]. Some editions of ROS come with visualization tools like Gazebo, while others require them to be installed externally. Moreover, ROS is distributed with a BSD (Berkeley Software Distribution) license that is known to be open source.

As shown in Table I, the first editions of ROS were developed in 2010, and the system has continued to evolve ever since, as a clear indication of the critical importance of the technology [10].

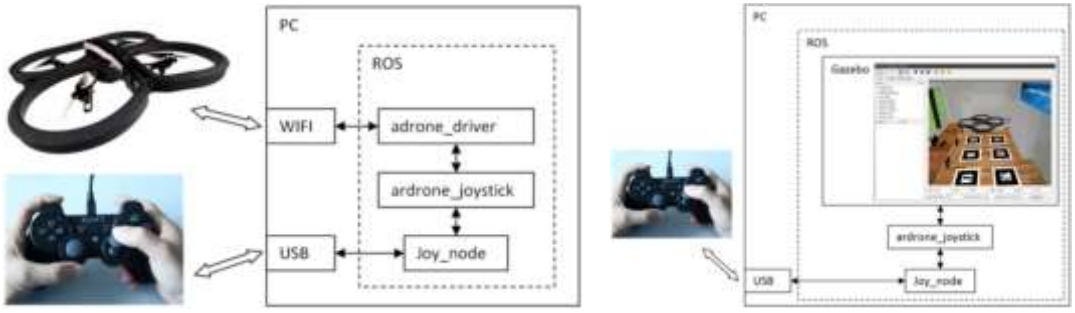
TABLE I
Ros version

ROS Version Name	Internal Visualizer	Visualizer Support	Release Date	Last Update
ROS Melodic Morenia	Gazebo 9.0	Gazebo 9.1	May 23rd, 2018	May, 2023
ROS Lunar Loggerhead			May 23rd, 2017	May, 2019
ROS Kinetic Kame			May 23rd, 2016	April, 2021
ROS Jade Turtle	gazebo 5.x	Gazebo 5.x - 6.x - 7.x	May 23rd, 2015	May, 2017
ROS Indigo Igloo	gazebo 2.x	Gazebo 2.x - 5.x	July 22nd, 2014	April, 2019
ROS Hydro Medusa			September 4th, 2013	May, 2015
ROS Groovy Galapagos			December 31, 2012	July, 2014
ROS Fuerte Turtle			April 23, 2012	--
ROS Electric Emys			August 30, 2011	--
ROS Diamondback			March 2, 2011	--
ROS C Turtle			August 2, 2010	--
ROS Box Turtle			March 2, 2010	--

B. Robot Control with ROS

ROS can be used as an interface between the physical robot and its user. For example, the user can control a physical robot connected to a ROS-installed PC by Wi-Fi through the use of a joystick connected to the same PC via USB (Fig. 2.a) [11].

ROS can also be used as an interface between a robot model developed in a simulator and the user, in which the user can control a virtual robot model developed in the Gazebo simulator using a joystick connected to a ROS-installed PC via USB (Fig. 2.b) [11].



a b

Fig. 2: Robot control with ROS

The general framework between ROS and Gazebo simulator is shown in Fig. 3.

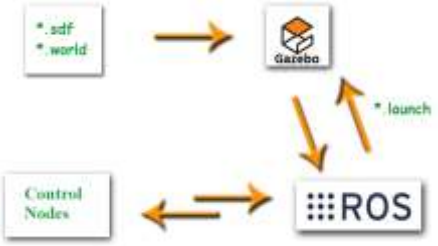


Fig. 3 ROS and Gazebo simulator

SDF (Simulation Description Format): SDF is a file format for the uploading or storage of information about a specific simulation setting or model in a Gazebo robot simulator. The developers of Gazebo designated this XML format for the description of objects and environments for the robot simulator. Models are defined using the SDF format, and the model can either be a simple figure or a complex robot. Models can also be used in a hierarchical structure. In other words, the structure of a model can be integrated with another model [12].

World Files: World files describe the environment file format created in the Gazebo robot simulator that typically require a "$ gazebo file_name.world" command at the terminal screen. There is no difference between the environment creation and model creation, and they can be formatted and operated using SDF [12].

Launch Files: Launch files are a file format that allows for the operation of several ROS Nodes, and that establish a link between ROS and a Gazebo

simulator. The message exchange between ROS and the simulator model is controlled by the nodes within the main launch files. They typically require a "$ roslaunch PacketName file_name.launch" command at the terminal screen [13].

ROS Topic: The topics are the channels that enable data transfer over the ROS network, and are based on the TCP/IP protocol. This protocol allows for the control of the model through the network.

IV. ROS-BASED CONTROL OF DJI MATRICE 100 ROBOT WITH QR CODES OBTAINED FROM DJI GUIDANCE

The control of a DJI Matrice 100 robot is realized using the QR codes shown in Fig. 4. For each ROS node to be controlled by the user, different QR codes were created, for example "M100 Engine Launcher", "M100 Autocontrol Launcher", "M100 TakeOff", "M100 Process Stopper", "M100 Landing" and "M100 Engine Stopper".

Fig. 4: QR codes used in the study

Within the scope of this study, a ROS node named "qr_controller" was created for the ROS-based control of a DJI Matrice 100 robot with QR codes obtained from DJI Guidance. The node includes the following phases: reception of QR codes through DJI Guidance, processing of QR images, operation or control of selected nodes.

A. Reception of QR Codes through DJI Guidance

The QR code in Fig. 4 represents the control command desired by the user, and was received by the right camera attached to the VBUS3 port of DJI Guidance using a software segment created in the C++ programming language that belongs to the "qr_controller" node, which is shown in Fig. 5.

```
Mat g_greyscale_image_right(240, 320, CV_8UC1);
void right_image_callback(const sensor_msgs::ImageConstPtr& right_img) {
   cv_bridge::CvImagePtr cv_ptr;
   try {
      cv_ptr = cv_bridge::toCvCopy(right_img, sensor_msgs::image_encodings::MONO8);
   }
   catch (cv_bridge::Exception& e) {
      ROS_ERROR("cv_bridge exception: %s", e.what());
      return;
   }
   g_greyscale_image_right= cv_ptr->image;
   cv::waitKey(1);
   Callback();
}
int main(int argc, char** argv) {
   ros::init(argc, argv, "qr_controller");
   ros::NodeHandle my_node;
   right_image_sub = my_node.subscribe("/guidance/right_image", 10, right_image_callback);
   while (ros::ok())
      ros::spinOnce();
   return 0;
}
```

Fig. 5: Reception of the image from the right camera attached to the VBUS3 port of DJI Guidance

The "right_image_callback" function shown in Fig. 5 launches the process of receiving the QR code from the right camera attached to the VBUS3 port of DJI Guidance. These codes were received at a frequency of 10 Hz and were designated to the 'cv::Mat'- type "g_greyscale_image_right" variable in order to be processed in the "qr_controller" node. The QR codes of the "g_greyscale_image_right" image were sent to the "ImageMatch" function shown in Fig. 6 for processing.

B. Processing of QR Images

In order to process the 'cv::Mat'- type "g_greyscale_image_right" image that was sent to the "ImageMatch" function in Fig. 6, the study used the "ZBar" package created by Paul Bovbel to read the QR codes.

```
#include <zbar.h>
string ImageMatch(cv::Mat P_image) {
   string snc;
   namedWindow("captured", CV_WINDOW_AUTOSIZE);
   ImageScanner scanner;
   scanner.set_config(ZBAR_NONE, ZBAR_CFG_ENABLE, 1);
   cv::Mat frame, frame_grayscale;
   frame=P_image;
   frame_grayscale=P_image;
   int width = frame_grayscale.cols;
   int height = frame_grayscale.rows;
   uchar *raw = (uchar *)(frame_grayscale.data);
   Image image(width, height, "Y800", raw, width * height);
   scanner.scan(image);
   int counter = 0;

   for (Image::SymbolIterator symbol = image.symbol_begin(); symbol != image.symbol_end(); ++symbol)
   {
    snc= symbol->get_data();
    if (symbol->get_location_size() == 4)
    {
       line(frame, Point(symbol->get_location_x(0), symbol->get_location_y(0)), Point(symbol->get_location_x(1), symbol-
>get_location_y(1)), Scalar(0, 255, 0), 2, 8, 0);
       line(frame, Point(symbol->get_location_x(1), symbol->get_location_y(1)), Point(symbol->get_location_x(2), symbol-
>get_location_y(2)), Scalar(0, 255, 0), 2, 8, 0);
       line(frame, Point(symbol->get_location_x(2), symbol->get_location_y(2)), Point(symbol->get_location_x(3), symbol-
>get_location_y(3)), Scalar(0, 255, 0), 2, 8, 0);
       line(frame, Point(symbol->get_location_x(3), symbol->get_location_y(3)), Point(symbol->get_location_x(0), symbol-
>get_location_y(0)), Scalar(0, 255, 0), 2, 8, 0);
    }
    counter++;
   }

   imshow("captured", frame);
   image.set_data(NULL, 0);
   waitKey(1);
   return snc;
}
```

Fig. 6: The function that performs the processing of QR images in the "ZBar" package

The "ImageMatch" function first determines the margins of the 'cv::Mat'- type "g_greyscale_image_right" image that it receives, and then reads the QR code by placing it into a frame, starting from the pixels in which the QR code is present (Fig. 7).

M100 Engine Launcher

M100 Engine Launcher with Border

M100 Landing

M100 Landing with Border

Fig. 7: Marking the QR codes in a "cv::Mat" - type image

The 'symbol->get_data()' - type data obtained after the processing of the framed QR code in the "ZBar" package were later designated to the string-type "snc" variable. It thus became possible for the textual equivalence of the QR code to be used along with the "snc" variable.

C. Operation or Control of Selected Nodes

The "Callback" function shown in Fig. 8 was called back inside the "right_image_callback" function shown in Fig. 5. This allowed for the operation of the "Callback" function at a frequency of 10 Hz, like the "right_image_callback" function.

```
void Callback(){
    if(ImageMatch(g_greyscale_image_right)=="M100 Engine Launcher")
    {
            obtain_control("True");
            Arm_Control("True");
    }
    else if(ImageMatch(g_greyscale_image_right)=="M100 Engine Stopper")
    {
            Arm_Control("False");
    }
    else if(ImageMatch(g_greyscale_image_right)=="M100 TakeOff")
    {
            if(!obtain_control("Control"))
                    obtain_control("True");
            takeoff_landing(dji_sdk::DroneTaskControl::Request::TASK_TAKEOFF);
    }
    else if(ImageMatch(g_greyscale_image_right)=="M100 Landing")
    {
            takeoff_landing(dji_sdk::DroneTaskControl::Request::TASK_LAND);
    }
    else if(ImageMatch(g_greyscale_image_right)=="M100 Process Stopper")
    {
            ...
            ...
}
```

Fig. 8: The function that was created for the operation or control of selected nodes

The "Callback" function called back at 10 Hz sends the 'cv::Mat' - type "g_greyscale_image_right" image obtained with the "g_greyscale_image_right" function to the "ImageMatch" function as a parameter, and the result of the "ImageMatch" function is controlled. This allows the operator to control the DJI Matrice 100 robot according to the QR code read by the right camera of DJI Guidance's VBUS3 port.

V. CONCLUSIONS

It is commonly observed in literature that the physical control of a DJI Matrice 100 robot platform can be realized using the Robot Operating System. The most serious problem encountered by researchers while working with the DJI Matrice 100 robot platform is the need to use a PC monitor to be able to run and control their software during the test phase of their field studies.

The findings of this study will provide researchers with savings in both time and costs during their studies, and will further contribute to literature in various ways when used in studies making use of the Robot Operating System.

ACKNOWLEDGMENT

This study was derived from Project No. 116E013, supported by the Scientific and Technological Research Council of Turkey (TUBITAK). We would like to express our gratitude to TUBITAK for its contribution to the study.

REFERENCES

[1] M. Iacono and A. Sgorbissa, "Path following and obstacle avoidance for an autonomous UAV using a depth camera," Robotics and Autonomous Systems, vol. 106, pp. 38-46, 2018.

[2] P. D. Nguyen, C. T. Recchiuto, and A. Sgorbissa, "Real-time path generation and obstacle avoidance for multirotors: a novel approach," Journal of Intelligent & Robotic Systems, vol. 89, no. 1-2, pp. 27-49, 2018.

[3] L. Teixeira, I. Alzugaray, and M. Chli, "Autonomous aerial inspection using visual-inertial robust localization and mapping," in Field and Service Robotics, 2018: Springer, Cham, pp. 191-204.

[4] T. Hinzmann, J. L. Schönberger, M. Pollefeys, and R. Siegwart, "Mapping on the fly: Real-time 3D dense reconstruction, digital surface map and incremental orthomosaic generation for unmanned aerial vehicles," in Field and Service Robotics, 2018: Springer, Cham, pp. 383-396.

[5] M100_User_Manual_EN. DJI Matrice 100 User Manual [Online] Available: https://dl.djicdn.com/downloads/m100/M100_User_Manual_EN.pdf.

[6] C. L. Doughty and K. C. Cavanaugh, "Mapping Coastal Wetland Biomass from High Resolution Unmanned Aerial Vehicle (UAV) Imagery," Remote Sensing, vol. 11, no. 5, p. 540, 2019.

[7] Y. H. Chai and A. K. Patil, "Inspired by Human Eye: Vestibular Ocular Reflex Based Gimbal Camera Movement to Minimize Viewpoint Changes," Symmetry, vol. 11, no. 1, p. 101, 2019.

[8] M. Quigley, E. Berger, and A. Y. Ng, "Stair: Hardware and software architecture," in AAAI 2007 Robotics Workshop, Vancouver, BC, 2007, pp. 31-37.

[9] Robotics_Simulator. Robotics Simulator [Online] Available: https://en.wikipedia.org/wiki/Robotics_simulator#cite_note-45.

[10] C. Dave. Ros Distributions [Online] Available: http://wiki.ros.org/Distributions

[11] H. Hongrong. tum_simulator [Online] Available: http://wiki.ros.org/tum_simulator.

[12] SDFormat. SDFormat [Online] Available: http://sdformat.org/spec

[13] N. Erik. Using rqt_console and roslaunch [Online] Available: http://wiki.ros.org/ROS/Tutorials/UsingRqtconsoleRoslaunch.