https://repository.uaeh.edu.mx/revistas/index.php/icbi/issue/archive
Pädi Boletín Científico de Ciencias Básicas e Ingenierías del ICBI

Pädi
ISSN: 2007-6363

Publicación Semestral Pädi Vol. 7 No. 14 (2020) 30–36

Line follower with a quadcopter
Seguidor de línea con un cuadricóptero.

David García-Olvera[a,*], Armando Hernández-Godínez[a], Benjamín Nicolás-Trinidad[a], Carlos Cuvas-Castillo[a]

[a]*Área Académica de Computación y Electrónica, Universidad Autónoma del Estado de Hidalgo, 42184, Pachuca, Hidalgo, México.*

Resumen

The present project consists in the implementation of a Parrot Bebop 2® quadcopter as an autonomous line follower. The development of the application is done through the operating system for robots ROS© , which, through scripts, establishes a communication link with Wi-Fi technology between the quadcopter and the computer that performs the line detection task and flight control. The script is written in Python 3.7 language using the Atom© text editor. The following features are integrated in the script: a control algorithm for the quadcopter flight, a line detector that operates with the video acquired from the quadcopter and a driver for the implementation of a joystick as a quadcopter manipulation method and security element.

Palabras Clave:
Line follower, Quadcopter, Python, ROS.

Abstract

El trabajo presentado consiste en la utilización de un cuadricóptero modelo Parrot Bebop 2® como seguidor de línea autónomo. El desarrollo de la aplicación se hace a través del sistema operativo para robots ROS© (Robot Operating System), el cual, con la elaboración de scripts. Establece un enlace de comunicación con tecnología Wi-Fi entre el cuadricóptero y la computadora que realiza la tarea de detección de línea y control de vuelo. El script está escrito en lenguaje Python 3.7 con uso del editor de texto Atom© . En el script se integran las siguientes características: un algoritmo de control para el vuelo del cuadricóptero, un detector de línea que opera con el video adquirido del cuadricóptero y un driver para la implementación de un joystick como método de manipulación del cuadricóptero y elemento de seguridad.

Keywords:
Cuadricóptero, Python, ROS, Seguidor de línea.

1. Introduction

Quadcopters are a very useful tools in areas like research, commercial or professional fields. These devices are used for different tasks like patrolling, army, monitoring, navigation, mapping and even advertising tasks, this is due to the versatility of its implementation (Piotr Kardasz and Zarzycki, 2016). The quadcopters can be programmated with routines that fulfill some activity, but in order to achieve this goal, a control algorithm is needed. This algorithm not only minimizes the error margin but also guaranties the security of its implementation under some circumstances, such like unexpected air currents (Johannes Meyer and von Stryk, 2012).

Figure 1: Parrot Bebop 2®.

A flight control algorithm (depending on the application and

[*]Autor para correspondencia: ga259965@uaeh.edu.mx

Correo electrónico: ga259965@uaeh.edu.mx (David García-Olvera), zack_armando@hotmail.com (Armando Hernández-Godínez), benjaminnt12@gmail.com (Benjamín Nicolás-Trinidad), carlos_cuvas@uaeh.edu.mx (Carlos Cuvas-Castillo)

Fecha de recepción: 05/08/2019 Fecha de aceptación: 25/10/2019 Fecha de publicación: 05/01/2020
https://doi.org/10.29057/icbi.v7i14.4727

purpose of the quadcopter) tend to be quite specific, in a line follower case, the quadcopter has to be able to follow a drawn line that indicates a fixed path to follow. There are different ways to do so, depending on how the quadcopter acquires information about its position. An example would be using the video camera integrated in the quadcopter, then, through digital image processing, it is obtained the relationship between the line to follow and some point of reference. The above example can be implemented specially in patrolling fields where the advantage of a wider observation field and a unmanned remote controlled devise tend to be more profitable (Alexandre S. Brandao and Soneguetti, 2015).

Conventionally, the line followers are limited to robots that move on the floor, which implement infrared sensors that detect the absence or existence of a line to follow (Pakdaman and Sanaatiyan, 2009) and with microcontrollers, operational amplifiers and mechanical switches do their job (M. Zafri Baharuddin and Chuan, 2016). This mention is made with the objective of contrasting the advantage of a quadcopter use, due that this one it's not attached to the floor; it can avoid obstacles and cover a wider field of movements for unforeseen events.

The quadcopter used in this work is made by Parrot® company. The Parrot Bebop 2® (Figure 1) model has a 14 megapixels HD 1080p video camera with a fisheye lens able to focus on 3 axes. It also has a digital stabilization system which is ideal for coping with winds up to 60 km/h and, in comparison to other models of the same company, its price is inexpensive (Parrot, 2016).

The Bebop Parrot 2® can be handled through different ways, the most popular is with smartphone applications that are developed officially by the manufacturer. In this work the manipulation of the quadcopter is done over the software development system for robots ROS© (Koubâa, 2017). This platform provides multiple open source tools for manipulation and control of this kind of devices. Currently, it has a driver called bebop_autonomy© for the Parrot bebop 1.0® and Parrot bebop 2.0® quadcopter models, which is based on the official SDK of the Parrot® company called ARDroneSDK3© (Bristeau et al., 2011). Bebop_autonomy© is open source and can be downloaded from the GitHub© platform.

The bebop_autonomy© driver uses the ROS© communication system, therefore, the control of the same is based on the elaboration of scripts that allows us to obtain and manipulate the information that the quadcopter throws through topics. Precisely, the video recorded is the most important feature of the work, since the information recorded on every frame is reinterpreted and processed in order to obtain an input to the control algorithm that has been developed (Monajjemi, 2015).

ROS© is only available for certain operating systems, the version used in this case was ROS Kinetic© (Koubâa, 2017) over the Ubuntu LTE 16.04© operating system. This platform works mainly with 2 programming languages: C ++ and Python. The elaborated scripts in this work were made in Python 3.7. One of the advantages of using scripts is the compatibility with the use of language libraries focused on digital image processing such as OpenCV© (Bradski and Kaehler, 2008) which is used in the elaboration of the work. Within the script, a driver for the control of the quadcopter through a joystick was added, in order to provide a method of manipulation of the quadcopter, in addition to granting a security control element in case there were any unforeseen event during the device flight.

The document is organized in sections as follows: Section 2 presents a general description of the implemented systems and some considerations concerned to the set-up of the experimental plant. Section 3 describes the structure of the script developed divided in 4 subsections: the communication system, joystick event reading, digital image processing and the implementation of the control algorithm. The paper ends with some conclusions.

2. System's structure

Figure 2 shows the system diagram. It represents the elements integrated as well as the relationship they have among, the inputs of the system are the video camera and the joystick gamepad. It is important to note that the video camera input has priority over the joystick input as far as piloting is concerned. However, the joystick keeps being a fundamental element since the node of communication is initialized by one of the buttons of the joystick, as well as some commands, like taking off and landing the quadcopter. The obtained outputs from the quadcopter are: the video camera focus position and the speed of the rotors for each helix that are given in 6 parameters divided into 2 categories: the linear speed and the angular speed (Figure 3), both categories are given in Cartesian coordinates in YAML format (for its acronym "YAML Ain't Markup Language"), which is a standard serialization of package information for programming languages (Ben-Kiki et al., 2005).

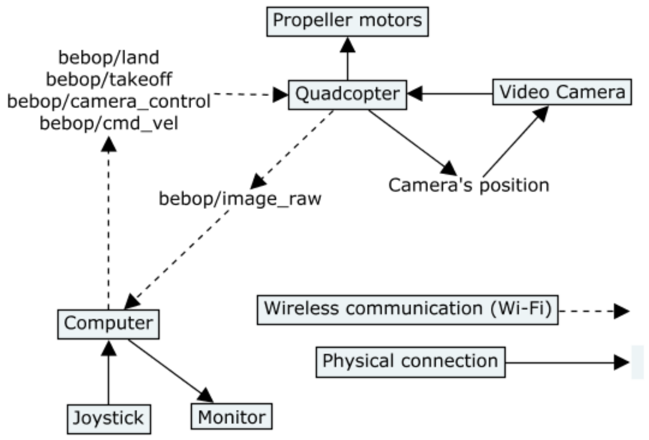


Figure 2: System diagram for the line follower.

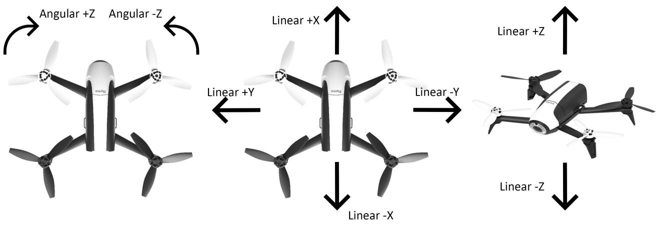


Figure 3: Bebop Parrot 2 navigation velocities.

2.1. Set-up and security elements

Along the performance and behavior testing of the quadcopter, the implementation of a mechanism to control and pilot the quadcopter was needed, in this case a XBOX 360® joystick was employed (Ikeda et al., 2012). The video recorded from the quadcopter, is visualized through an Ubuntu's environment window, which is a direct reading of the quadcopter's bebop/image_raw topic. All tests were performed in isolated environments in order to safeguard the physical integrity of third parties, as well as the integrity of the quadcopter to unforeseen events, such as air currents that exceed the quadcopter's self-stabilization capacity. For testing, a line of approximately 25 meters was drawn with blue tape on a surface without unevenness.

3. Script structure

The script consists of 3 features. The first one is the method of communication of ROS© , this allows communication between the ROS© core and the quadcopter, enabling the reading and writing of topics. The second feature is the manipulation of the quadcopter with the joystick, in this part the script identifies the characteristics of the joystick, as well as its input parameters; which are interpolated through a function to be published in the topics of movement speed and camera position of the quadcopter. The script also assigns the quadcopter's take-off, landing and camera initialization commands to the joystick buttons. Although this can be done with command lines, the immediate response of the joystick provides an extra security measure for the piloting of the quadcopter. Finally, the third feature is the digital image processing of the video recorded by the quadcopter and the implementation of the control algorithm. In this part, the image is obtained from the quadcopter's /image_raw topic, then, the image is identified and manipulated by the script in order to process the information about the line to follow and make decisions that handle the movement speed of the quadcopter.

3.1. ROS© communication system

The script was written with the qualities of a node within the core of ROS© . To achieve this goal is necessary the use of the rospy library that is part of ROS© repositories (Quigley et al., 2015). Then, the script is able to interact with topics that are available within the ROS© core at the moment. Figure 4 shows the initialization of the script as a ROS© node, Figure 5 shows the publisher metodology with 4 different objective topics which are: the topic of landing, taking off, camera control and movement speed of the quadcopter. These four topics are the only ones to work with as far as publication is concerned. Figure 6 shows the initialization of the subscriber to the video camera topic of the quadcopter, this element is a fundamental part since it is responsible of obtaining the image of the quadcopter.

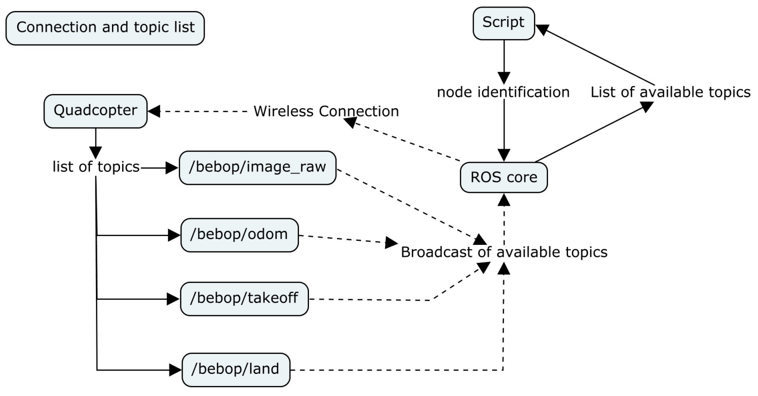


Figure 4: Node identification and topic listing.

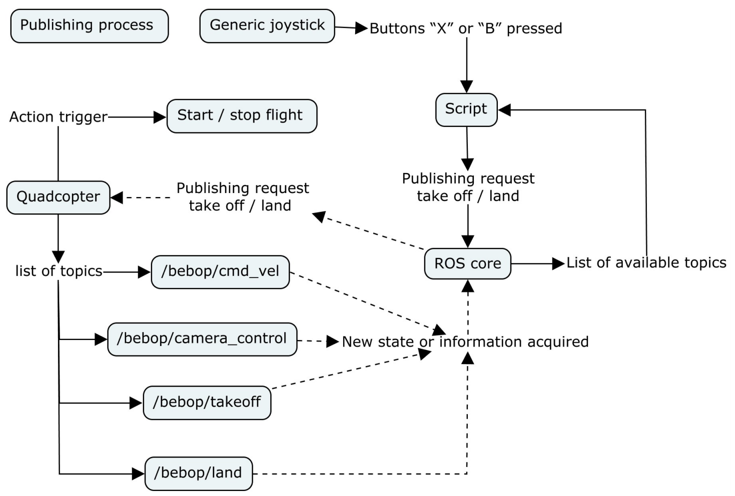


Figure 5: Publishing process within the system.

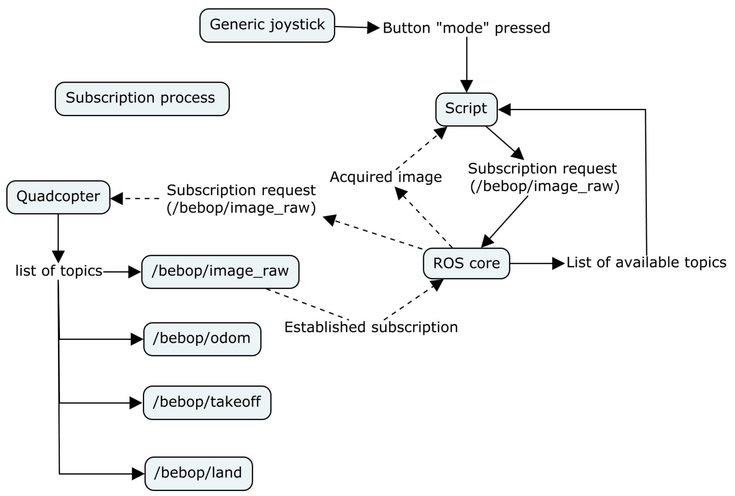


Figure 6: Subscribing process within the system.

3.2. Joystick event reading

The script recognizes the joystick device by reading a file within the Ubuntu© operating system. That is hosted at /dev/input directory address (Nguyen, 2003) and consists of a structure that records the events performed on the joystick. The ioctl function of the fcntl (Stevens et al., 2008) library of the repository of Unix© (W. Richard Stevens and Rudoff, 2004) based operating systems, allows the reading and recognition of the device through responses generated by the operations defined by the ioctl function. With this function the name of the device and the number and mapping of buttons and axis that the joystick has are obtained. Both, the axis and the buttons have an alias in hexadecimal number, for practicality these numbers are stored in a dictionary within the script. The dictionary stores the hexadecimal numbers for the buttons and axis of the most common generic controls that can be implemented within the operational system (DAI and SHU, 2008).

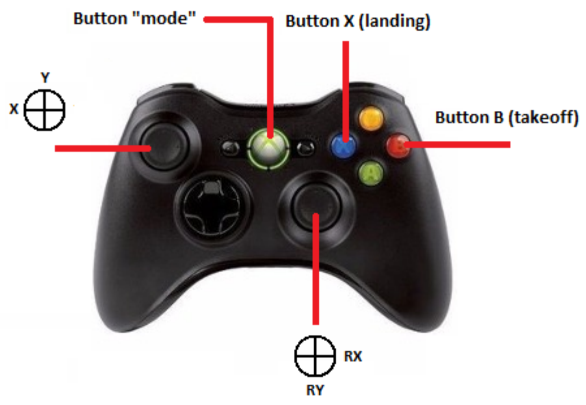


Figure 7: Assignment of buttons and axis map of the joystick.

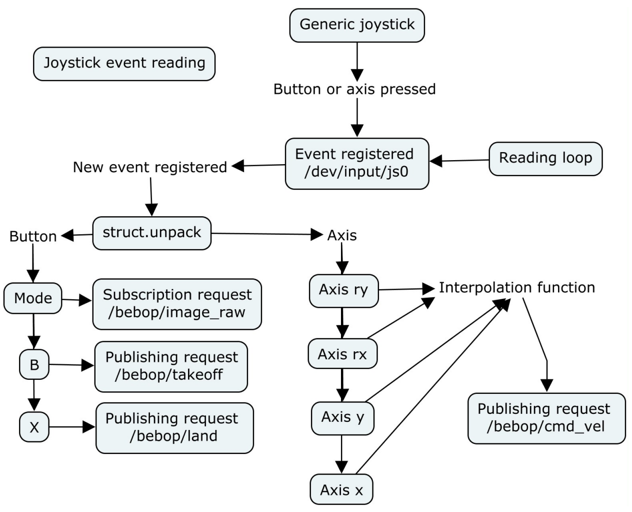
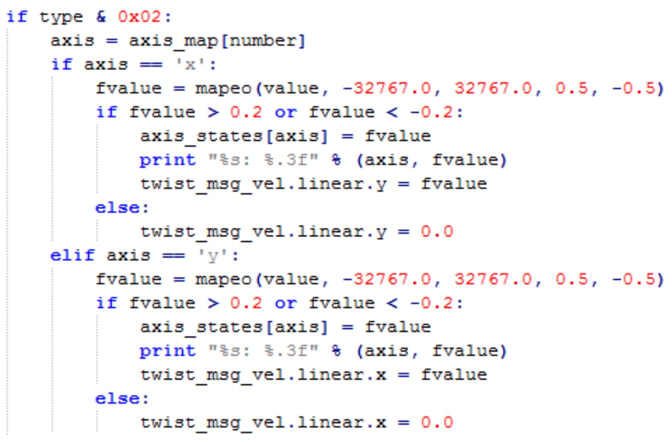


Figure 8: Joystick event reading diagram.

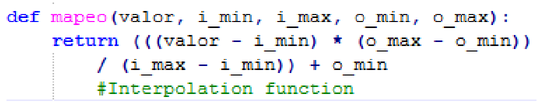
Each time that there's an event (pressing a button or moving an axis), the script reads and interprets the information corresponding to it, based on 4 parameters which are: time, value, type and number. Time corresponds to the moment in which the event occurred, value identifies in the case of the axis the stick position in a range of -32767 to 32767 and, in the case of buttons, 1 if it is pressed and 0 if it is not. Type identifies what kind of event it was, if it was from the axis, button or initialization type. The initialization ones show information of the joystick status at the first moment of running the driver. Finally the number identifies which button or axis were changed (DAI and SHU, 2008).



```
if type & 0x02:
    axis = axis_map[number]
    if axis == 'x':
        fvalue = mapeo(value, -32767.0, 32767.0, 0.5, -0.5)
        if fvalue > 0.2 or fvalue < -0.2:
            axis_states[axis] = fvalue
            print "%s: %.3f" % (axis, fvalue)
            twist_msg_vel.linear.y = fvalue
        else:
            twist_msg_vel.linear.y = 0.0
    elif axis == 'y':
        fvalue = mapeo(value, -32767.0, 32767.0, 0.5, -0.5)
        if fvalue > 0.2 or fvalue < -0.2:
            axis_states[axis] = fvalue
            print "%s: %.3f" % (axis, fvalue)
            twist_msg_vel.linear.x = fvalue
        else:
            twist_msg_vel.linear.x = 0.0
```

Figure 9: Axis reading and interpolation function implementation.

Three buttons were assigned to control the quadcopter (Figure 7): the "X" button publishes to the landing topic, the "B" button publishes to the takeoff topic and the "mode" button starts the subscription to the video camera as well as the image processing and control algorithm of the quadcopter (Figure 8). If the "mode" button has not been pressed the quadcopter can be piloted with the following axis: the x axis controls the linear speed at y, the y axis controls the linear speed at x, the rx axis controls the angular velocity at z and the axis ry controls the linear velocity at z (Figure 9). It is important to mention that there is a previous interpolation between the values obtained by the joystick controller and the range allowed for the publication of the quadcopter's operating speeds, this is done with a function of interpolation that returns the conditioned value (Figure 10).



```
def mapeo(valor, i_min, i_max, o_min, o_max):
    return (((valor - i_min) * (o_max - o_min))
        / (i_max - i_min)) + o_min
        #Interpolation function
```

Figure 10: Definition of the interpolation function.

3.3. *Digital image processing*

The control algorithm begins at the moment when the "mode" button is pressed. The button starts the node subscription to the /bebop/image_raw topic. At the same time, this instruction uses a "callback" function where the image processing takes place. OpenCV© offers different tools as far as image processing is concerned, with the help of the CVBridge function, an exception method is defined, this method allows to notify if there is an event that prevents the execution of an OpenCV© function. In order to process the image obtained from the /bebop/image_raw topic, it must be encoded from the message format of ROS© to an OpenCV© object, this is achieved with the CVBridge imgmsg_to_cv2 function (Martinez and Fernández, 2013).

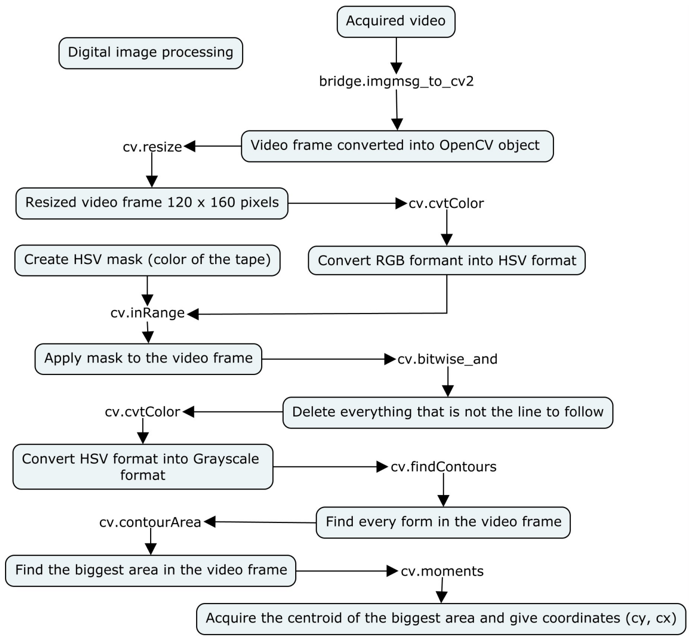

Digital image processing
Acquired video
bridge.imgmsg_to_cv2
Video frame converted into OpenCV object
cv.resize
Resized video frame 120 x 160 pixels
cv.cvtColor
Create HSV mask (color of the tape)
Convert RGB formant into HSV format
cv.inRange
Apply mask to the video frame
cv.bitwise_and
Delete everything that is not the line to follow
cv.cvtColor
Convert HSV format into Grayscale format
cv.findContours
Find every form in the video frame
cv.contourArea
Find the biggest area in the video frame
cv.moments
Acquire the centroid of the biggest area and give coordinates (cy, cx)

Figure 11: Video camera image conditioning diagram.

The default setting adjusts the quadcopter's camera to capture the front area of itself, so, to visualize the ground where the line to be detected is located, the speed parameter -83 has to be published on the /bebop/camera_control topic. This modifies the angular "y" camera position and consequently the area of visualization. This is only done once and preferably it is not modified later. The camera has a position stabilizer which can modify the position of the camera, therefore if the instruction to visualize towards the ground is constant, it is expected that the stabilizer will not move the camera to another unwanted position.

By default, the size of the image obtained is 1920 x 1080 pixels, this resolution is not suitable for image processing, since the computation time per frame is significantly larger, so, the image is resized with a resolution of 160 x 120 pixels. Another of the libraries used in the process is NumPy© , this library is used for the manipulation and creation of matrices. The video recorded, after being converted into an OpenCV© object, becomes a three-dimensional arrangement that can be interpreted by NumPy© (Oliphant, 2006).

There are different color detection algorithms, the HSV (Hue-Saturation-Value) model is ideal for the detection of colors, since a range can be established between the hue of the color to be detected, despite the change of saturation or brightness, it will remain the same (Deswal1 and Sharma, 2014). The line to be detected is marked by a blue tape. To determine the range in HSV parameters of the blue color of the tape itself, it was necessary to take samples of the blue tape with a camera at different light exposures, then convert the RGB (Red-Green-Blue) values into HSV parameters.

With the function cv.inRange a matrix mask is obtained, it contains the information corresponding to the blue range of colors of the line to be detected. Then, with cv.bitwise_and function, it's extracted the blue line to follow from every frame of the video recorded by the quadcopter's camera. The obtained video has only the range of color established by the mask previously created for the blue of the tape in a wide range of brightness (Figure 12).

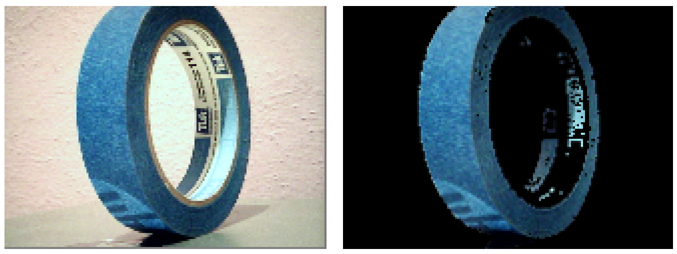


Figure 12: Exclusive detection of the blue color of the tape.

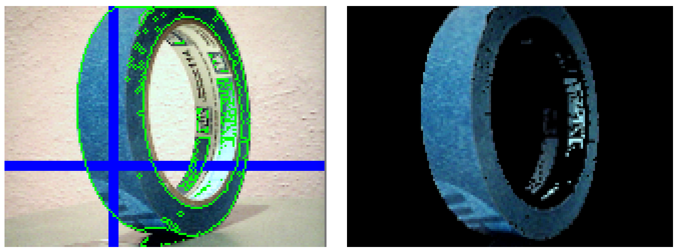


Figure 13: Obtaining the contour and centroid of the blue figure.

The video obtained from the mask is converted to gray scale, then the OpenCV© function (cv.findContours) allows to identify the contour of the tape of the line to follow. If there is any contour in the video, it means that the line exists; the way in which the noise detection of the image is reduced, is to take the larger detectable area. So, if there is any element that is not the line to follow this becomes discarded unless it has a larger area than the line to follow itself (García et al., 2015).

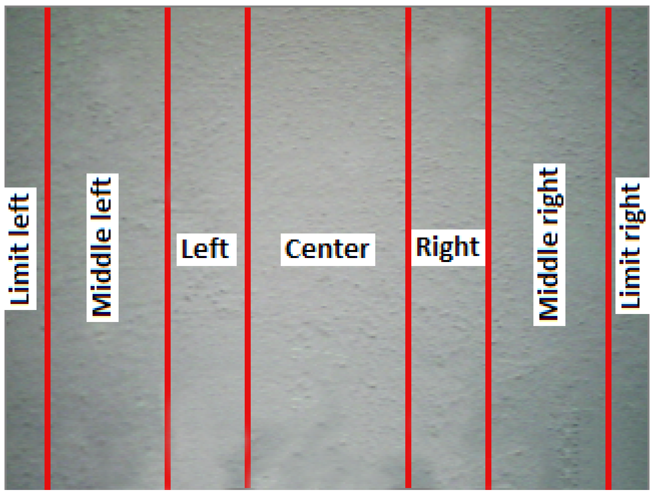


Figure 14: Definition of areas in the main video.

Cv.moments function determines the centroid of the line to follow as well as its coordinates in the video, which are stored in the variables cx (horizontal position) and cy (vertical position). These coordinates are given in a range defined by the resolution of the processed image, which in this case is 120 (cy) x 160 (cx) pixels. The centroid is the reference point for the quadcopter piloting, its position determines in which direction the quadcopter must move (Figure 13).

Any movement of the quadcopter in any direction will change the position of the centroid of the line to follow, then the quadcopter must be able to return to the point where the centroid of the line to follow is right at the center of the video, in order to always have the line visible. This is carried out with the determination of areas throughout the video as seen in the Figure 14. Six vertical red lines in the video determines different areas for possible cases where the line to follow can be detected as well as the decisions regarding the control of the quadcopter.

Table 1: Relationship between movement speeds and the designed areas in the video recorded.

	Pixel range	Angular “z” speed	Linear “x” speed	Linear “y” speed
Center	cx <100 cx >60	0.0	0.05	0.0
Left	cx <= 120 cx >= 100	-0.15	0.025	-0.1
Middle left	cx <150 cx >120	-0.3	0.025	-0.25
Limit left	cx >= 150	-0.7	0.0	0.0
Right	cx <= 60 cx >= 40	0.15	0.025	0.1
Middle right	cx <40 cx >10	0.3	0.025	0.25
Limit right	cx <= 10	0.7	0.0	0.0
Lineless		0.0	0.0	0.0

3.4. *Control algorithm*

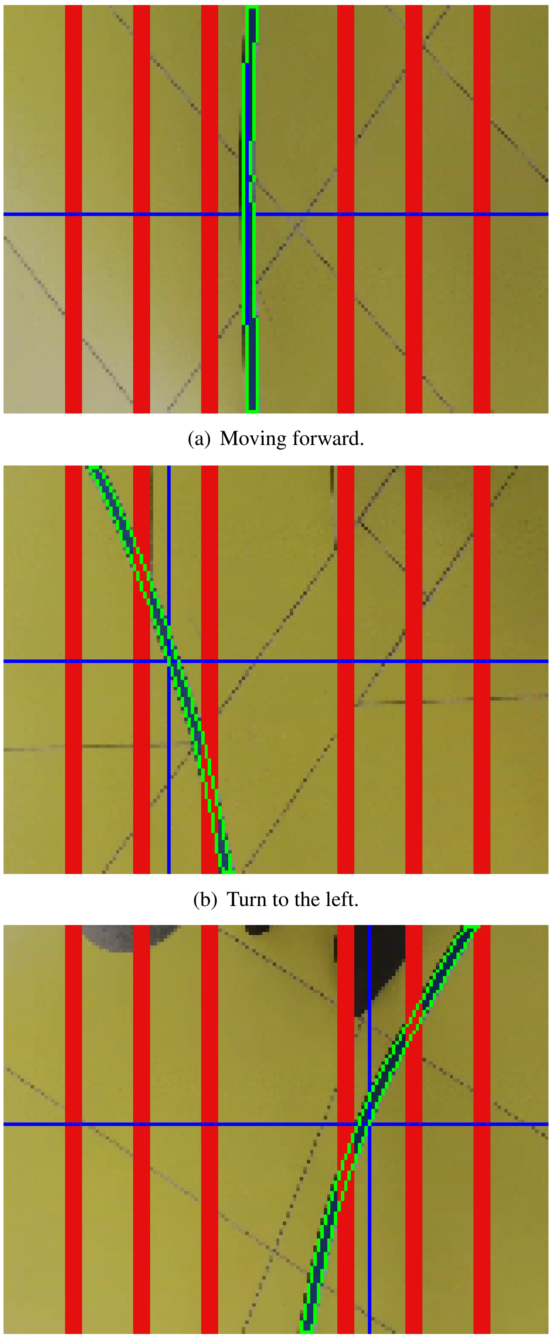
The six red lines drawn in the quadcopter video determine 7 areas. These lines are distributed according to the width in pixels of the processed video which in this case is 160 pixels, each determined area obeys different decisions by the quadcopter if the centroid of the line detected is found. This is interpreted as different movement speeds what will try to head the quadcopter to the central area. In addition, in the case where there is no line to follow found in the video, the quadcopter stops moving. Table 1 shows the relationship between the published speeds to the topic /bebop/cmd_vel and the area where is located the centroid of the line to follow. It is important to emphasize that the cx coordinate is a parameter given by the centroid of the detected line and it is fixed in a range of 0 to 160 as well as the width of every video frame.

Finally, Figure 15 shows the detection result and how the quadcopter moves in 3 cases:

- Straight line (Figure 15(a)): the centroid of the line to follow is between 60 and 100 pixels with respect to the vertical axis (cx) of the image (central area), the quadcopter moves forward only.

- Turn to the left (Figure 15(b)): the centroid of the line to follow is between 120 and 100 pixels with respect to the vertical axis (cx) of the image (left area). The quadcopter moves slightly to the left modifying both the angular velocity on the z axis, and the linear velocity on the y axis. Also the linear velocity in x decreases by half to make a better turn without stop the movement on the line to follow.

- Turn to the right (Figure 15(c)): the centroid of the line to follow is between 60 and 40 pixels with respect to the vertical axis (cx) of the image (right area). As in the left turn the quadcopter moves slightly to the right, keeping the same values for the angular and linear angular velocities but with the opposite sign. The linear movement velocity in x also decreases but does not change its sign (reverse).


(a) Moving forward.

(b) Turn to the left.

(c) Turn to the right.

Figure 15: Quadcopter shift in different scenarios.

4. Conclusions

Part of the application precision comes from the detection of the line, where it matters the size of the tape, the distance between the quadcopter to the floor, the color detection algorithm, the movement speed of the quadcopter and the acquisition speed from the video camera.

During the development of this work, it was considered in first instance to use a black tape for the line, however the techniques used for detection of this color failed under the effects of exposure to different levels of brightness, hence, the use of a black tape was discarded, concluding that the use of the HSV format is more useful to detect ranges of any other color.

The control algorithm does not implement any mathematical model, the proposed movement speeds are the product of the observation by the work developers.

The system can be improved implementing a more efficient control stage by using the same principle of line color, coordinates and centroid detection.

The work developed achieved its objective as a line follower, however, the results presented here can be used for different tasks where tracking requirements are implemented.

References

Alexandre S. Brandao, F. N. M., Soneguetti, H. B., 2015. A vision-based line following strategy for an autonomous uav. 12th International Conference on Informatics in Control, Automation and Robotics 2, 317–318.
DOI: 10.13140/RG.2.1.1132.2729

Ben-Kiki, O., Evans, C., Ingerson, B., 2005. Yaml ain't markup language (yaml) version 1.1. yaml. org, Tech. Rep, 23.

Bradski, G., Kaehler, A., 2008. Learning OpenCV: Computer vision with the OpenCV library. .O'Reilly Media, Inc.".

Bristeau, P.-J., Callou, F., Vissiere, D., Petit, N., 2011. The navigation and control technology inside the ar. drone micro uav. IFAC Proceedings Volumes 44 (1), 1477–1484.

DAI, Y.-f., SHU, W.-q., 2008. Linux-based implementation of game joysticks. Computer Engineering & Science (7), 36.

Deswal1, M., Sharma, N., 2014. A fast hsv image color and texture detection and image conversion algorithm. International Journal of Science and Research (IJSR) 3, 5.

García, G. B., Suarez, O. D., Aranda, J. L. E., Tercero, J. S., Gracia, I. S., Enano, N. V., 2015. Learning image processing with opencv. Packt Publishing Ltd.

Ikeda, J., Ledbetter, C. L., Bristol, P., Apr. 24 2012. Game controller with multipositional state controller element. US Patent App. 29/368,839.

Johannes Meyer, Alexander Sendobry, S. K. U. K., von Stryk, O., 2012. Comprehensive simulation of quadrotor uavs using ros and gazebo. Simulation, Modeling, and Programming for Autonomous Robots: Third International Conference 1, 1.
DOI: 10.1007/978-3-642-34327-8_36

Koubâa, A., 2017. Robot Operating System (ROS). Springer.

M. Zafri Baharuddin, Izham Z. Abidin, S. S. K. M. Y. K. S., Chuan, J. T. T., 2016. Analysis of line sensor configuration for the advanced line follower robot, 1–2.

Martinez, A., Fernández, E., 2013. Learning ROS for robotics programming. Packt Publishing Ltd.

Monajjemi, M., 2015. bebop_autonomy-ros driver for parrot bebop drone (quadrocopter) 1.0 & 2.0. Recuperado el 14.

Nguyen, B., 2003. Linux Filesystem Hierarchy. Binh Nguyen.

Oliphant, T. E., 2006. A guide to NumPy. Vol. 1. Trelgol Publishing USA.

Pakdaman, M., Sanaatiyan, M. M., 2009. Design and implementation of line follower robot. IEEE 43, 1–2.
DOI: 10.1109/ICCEE.2009.43

Parrot, S., 2016. Parrot bebop 2. Retrieved from Parrot. com:http://www.parrot.com/products/bebop2.

Piotr Kardasz, Jacek Doskocz, M. H. P. W., Zarzycki, H., 2016. Drones and possibilities of their using. Journal of Civil & Environmental Engineering 6, 5–6.
DOI: 10.4172/2165-784X.1000233

Quigley, M., Gerkey, B., Smart, W. D., 2015. Programming Robots with ROS: a practical introduction to the Robot Operating System. .O'Reilly Media, Inc.".

Stevens, W. R., Fenner, B., Rudoff, A. M., 2008. UNIX Network Programming: The Socets Network API. Addison-Wesley.

W. Richard Stevens, B. F., Rudoff, A. M., 2004. Addison-Wesley, Boston, MA 02116, USA.