A pipeline framework for robot maze navigation using computer vision, path planning and communication protocols.

RODRIGUEZ-TIRADO, A., MAGALLAN-RAMIREZ, D., MARTINEZ-AGUILAR, J.D., MORENO-GARCIA, C.F., BALDERAS, D. and LOPEZ-CAUDANA, E.

2020

A pipeline framework for robot maze navigation using computer vision, path planning and communication protocols

Areli Rodriguez-Tirado
Tecnologico de Monterrey,
School of Engineering and Sciences
Mexico City, Mexico
a01337316@itesm.mx

Daniela Magallan-Ramirez
Tecnologico de Monterrey,
School of Engineering and Sciences
Mexico City, Mexico
a01652247@itesm.mx

Jorge David Martinez-Aguilar
Tecnologico de Monterrey,
School of Engineering and Sciences
Mexico City, Mexico
a01337729@itesm.mx

Carlos Francisco Moreno-Garcia
Robert Gordon University
School of Computing
Aberdeen, UK

David Balderas
Tecnologico de Monterrey,
School of Engineering and Sciences
Mexico City, Mexico

Edgar Lopez-Caudana
Tecnologico de Monterrey,
School of Engineering and Sciences
Mexico City, Mexico

***Abstract*—Maze navigation is a recurring challenge in robotics competitions, where the aim is to design a strategy for one or several entities to traverse the optimal path in a fast and efficient way. To do so, numerous alternatives exist, relying on different sensing systems. Recently, camera-based approaches are becoming increasingly popular to address this scenario due to their reliability and given the possibility of migrating the resulting technologies to other application areas, mostly related to human-robot interaction. The aim of this paper is to present a pipeline methodology towards enabling a robot solving maze autonomously, by means of computer vision and path planning. Afterwards, the robot is capable of communicating the learned experience to a second robot, which then will solve the same challenge considering its own mechanical characteristics which may differ from the first robot. The pipeline is divided into four steps: (1) camera calibration (2) maze mapping (3) path planning and (4) communication. Experimental validation shows the efficiency of each step towards building this pipeline.**

***Index Terms*—Robot navigation, computer vision, camera calibration, mapping, path planning, communication, NAO robot, educational innovation, higher education.**

I. Introduction

In robot navigation, an autonomous system has the ability to take real-time decisions based on the problem that is presented to it. Typically, by means of different sensing options and data processing algorithms, the robot is made aware of its environment. Nonetheless, most recent approaches rely solely on the images obtained from the robot's camera. This has been the most widely used choice in competitions and practical applications [1], not only since the robot is not limited by the characteristics other sensors (e.g. infrared, light) which derive on conditional programming scenarios (i.e. an issue which derives on technical limitations of robot components and the integrated sensors which are different for each platform), but also due to the capability to transfer these methods towards other real-life applications, such as education, industry 4.0, healthcare, among others.

In this paper, we present a pipeline framework that enables a NAO robot[1] to traverse a maze in an autonomous way. To achieve this, we focused in the integration of four main stages: (1) camera calibration, which is useful for adjusting camera parameters on different platforms; (2) mapping, which includes vision algorithms to process images and generate an internal map; (3) path planning, to navigate the maze and make decisions about the optimal way out; and (4) communication, to transmit knowledge and make it possible for a second robot to solve the maze faster and considering its own characteristics.

NAO are humanoid robots developed in 2008 by Softbank Robotics. They can be programmed using a multi-platform application called *Choregraphe* or programming languages such as C++ or Python. Furthermore, the robots have a wide variety of sensors, including seven touch sensors, omni-directional microphones, ultrasonic sensors, and 2D cameras that allow them to interact with their surroundings[2]. NAO robots are now in their 6^{th} version and have become a staple in education and research alike.

II. Related Work

Computer vision has several implementations in robotics. For instance, Respall et al. [2] applied it in quadcopters. This project was focused on tracking an unmanned ground vehicle (UGV) platform by getting color information of the image. Base on the data the position of the UGV platform was determined and the quadcopter would follow it. This article proposes to use vision algorithms to reconstruct a path or

[1]https://www.softbankrobotics.com/emea/en/nao
[2]http://doc.aldebaran.com/2-1/family/index.html

maze as the robot moves through it. This will allow to use path planning algorithms so that the robot can make the best decision while navigating.

Cortés et al. [1] presented a system where a trio of robots guided a crowd through the corridors of an airport. This system used the cameras to estimate the pose of all entities, so that the robots could create a formation that ensured that all participants where enclosed within and did not get lost. In addition, the system allows a human to perform minor corrections on the automated mappings to further enhance the pose estimation. The method proved to be effective when the robots work at the same time and with the same characteristics but does not offer the possibility of transferring the knowledge between entities or to consider different mechanical characteristics.

Kumar et al. [3] worked with path planning and avoiding obstacles algorithms to solve certain routes. By using a linear regression method, a NAO robot was trained whit 500 different scenarios. In a second part of the work was involved a second robot with the same algorithm. The main purpose was that both robots to cross without any collision. The obstacles consisted in cylinders and the robot had to follow a establish path, so it just had to get off one corner to the other, with a known trajectory to be adjust depending of physical obstructions.

Schranz et al. [4] this article mention advantage of use communication between robot for certain activities as navigation. The communication architecture used are called swarm, which is a non centralized communication. Proposal that this type of communication is used for systems that need to adapt to their environment.

III. Methodology

Each of the steps in the pipeline were developed separately; however, the flow chart in Figure 1 shows their interconnection. In this section, we describe the technologies used for each stage.

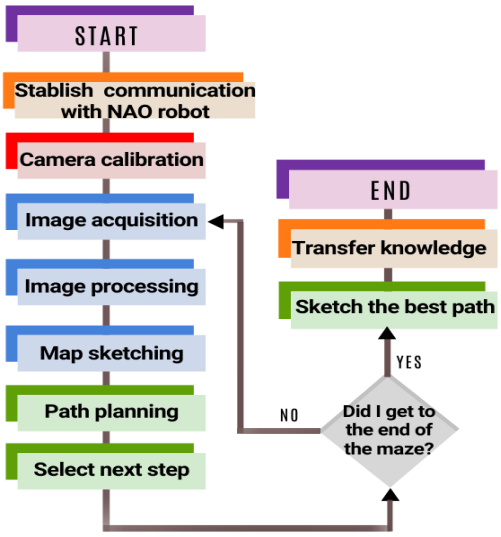


Fig. 1: Pipeline of the proposed system.

A. Camera calibration

Firstly, the camera must be calibrated so that the captured images are not distorted and the results obtained from the computer vision algorithms are less prone to errors. We have implemented a method presented by Zhang [5] to generate a calibration matrix that helps correct these distortions by taking multiple images of a well-defined object, in this case, a checkerboard. Beforehand, the camera parameters (brightness, saturation, focal length, etc.) must be manually configured to obtain the best possible image. Afterwards, a matrix is generated based on the corner patterns of the checkerboard. We used a pattern of 6×5 corners, as shown in Figure 2. Afterwards, we detect the corners using the *cv2.cornerSubPix* function in *OpenCV* for Python. We then calculate the camera matrix and its distortion coefficients using *cv2.calibrateCamera*. Finally, with the obtained information we generate a new matrix that is used to undistort the images.

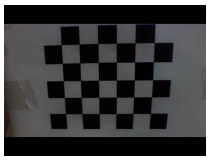


Fig. 2: Chessboard image used for calibration

B. Mapping

Using Coppelia Sim software[3], a 3D model of a maze was deployed in order to record a video simulating the robot traversing it. The methodology described below was applied to each frame of said video in order to sketch the labyrinth and obtain an internal map for the robot.

The first step consists on applying the Canny algorithm to divide the image into different areas which allow to better control what the robot detects and maps. Three zones are obtained (1) the top zone (i.e. the farthest), (2) the middle zone and (3) the bottom zone (i.e. the closest). The robot uses the middle zone to sketch the maze, and anything above or beyond zones 1 and 3 is discarded, Afterwards, the Hough transform [4], [6], followed by a merge algorithm[5], are applied to each zone in order to find the lines which define the maze. Then, the obtained middle lines are superimposed over a black image using a perspective transform[6]. This allows a better perspective of the space in front the robot, allowing it to determine approximate distances from the surrounding environment. For the estimation of distances, we used the Euclidean distances based on the pixels.

Once the distances from the robot to the different points are calculated, in a new black image the map sketching takes place by using a previously defined starting point and then taking the distances to a $10:1$ ratio, so that the entire map fits into a

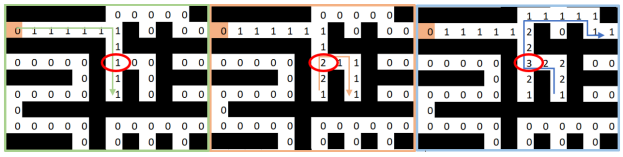
[3]https://www.coppeliarobotics.com/helpFiles/index.html
[4]https://patents.google.com/patent/US3069654
[5]https://stackoverflow.com/questions/45531074/how-to-merge-lines-after-houghlinesp
[6]https://github.com/ndrplz/self-driving-car

single image. The lateral lines that the robot visualizes in each frame were drawn from the starting point. When the sketching for this frame ends, the starting point is updated, taking as the new starting point the last coordinate of the drawn lines. In this way, the next segment of the map will be sketched just after what has already been plotted.

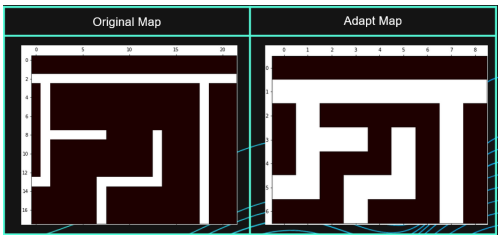
C. Path planning

The Tremaux algorithm is one of the most efficient methods to get out of a maze [7]. It consists on marking all the steps and the direction that a maze-solving entity comes from. Using this method, repeating the traversing of a path can be avoided. For this implementation, we established priorities for the following four moves: (1) front step, (2) left step, (3) right step and (4) back step. Moreover, we implemented an exception on the Tremaux algorithm rule of never returning to a node twice since this could lead to the robot getting caught in a situation such as the one shown in Figure 3a. Still, we maintain the rule which prioritizes the nodes with less visits.

Naturally, a maze image may contain either false walls or paths which can disrupt the path planning process. Therefore, the algorithm considers that each pixel depicting the walls and paths may have a possible error range. Thus, we implemented a maze adjustment as shown in Figure 3b prior to executing the path planning phase.



(a) Image representing a robot passing three times through the same node.



(b) Maze adjustment.

Fig. 3

D. Communication

A NAO robot has two communications methods with a computer in order to be programmed: Ethernet and Wi-Fi[7]. The latter was selected since the robot will be in constant motion. Currently, any version of a NAO robot can be programmed using 32-bits Python 2.7[8].

Web services such as Ubidots, Firebase and Google Drive API can be implemented for communication purposes using different programming languages, including Python and considering the JSON standard. Ubidots has the advantage of a graphical interface or dashboard that allows to monitor discrete values sent via a widget, with a maximum of four write or read requests every second[9]. Firebase lacks this functionality but offers the possibility to store up to 1 GB and to transfer 20,000 write requests and 50,000 read requests per day[10]. Regarding Google Drive API, the storage limit depends on the user's account, and its advantage in comparison to the previous approaches is that any type of file can be stored. Up to 50,000 Google Drive write requests per day can be performed[11].

Figure 4 shows a diagram of the operation of the proposed communication module. Two robots, named Curie and Atom, act as master and slave respectively. On a central computer, the programming code for Curie's motion controls acquired, stored and compiled in Python 2.7 by means of the image obtained from the upper camera. Moreover, the communication protocol is generated using the JSON standard with Ubidots. This code is then uploaded to Curie's CPU via the ALProxy function, found inside the NAO Python SDK NAOqi, which creates a proxy server with the robot. The image stored on the computer is retrieved by another script which processes the Canny function[12] (implemented in Python 3.3.7). On the other hand, Atom compiles a second Python 2.7 based code on its CPU, which ensures that a value stored in a global variable is called in Ubidots through a second JSON-based script. This value is a positive integer that Atom interprets to make a specific move.

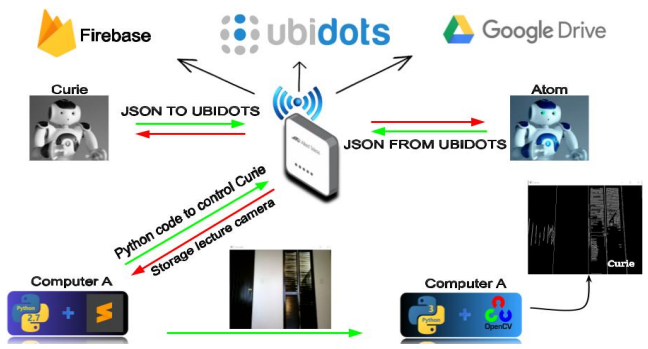


Fig. 4: Communication module operation diagram.

IV. Test Bench

A. Camera calibration

We took several pictures of a checkerboard in different angles and under different light conditions. It is important to mention that calibration focuses on a region of interest (ROI). In our case, this area was the checkerboard itself. We realized that an increase in the number of images yielded better results,

[7]http://doc.aldebaran.com/2-1/family/robots/connectivity_nao.html
[8]http://doc.aldebaran.com/2-5/dev/python/install_guide.html
[9]https://ubidots.com/
[10]https://firebase.google.com/
[11]https://developers.google.com/analytics/devguides/config/mgmt/v3/limits-quotas
[12]https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html

however we decided to stop at 51 images since there were no significant changes after that.

B. Mapping

During the testing phase, the possibility of using a corner detection algorithm was raised; however, the algorithm was detecting corners of the rendered image, and so in a real situation with smoother images, those characteristics would not be present and therefore, the code would need to be re-adapted to perform in a real environment. Also, we did not want to limit the code to a maze with corners, since this would limit its adaptability to other environments. For these reasons, this option was discarded in favor of line detection.

Another important test aspect was to define the values for the Canny function since, as mentioned in [6], the performance of Hough transform depends on the image that the Canny algorithm provides. After several tests, it was determined that to clean the image and keep only the important lines, a second Gaussian filter should be applied in addition to the first Gaussian one. The filter didn't affect lines detection.

The most challenging stage, and the one in which most tests were carried out, was in the sketch of the internal map, as there were issues with the turns. Figure 5 (left) shows how the sequence of the hallways was lost at these points. In contrast, Figure 5 (right) shows that it was possible to fix this by changing the condition of the adjustment for the new center and stop mapping during the laps.

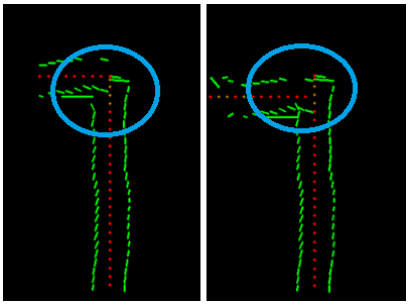


Fig. 5: Corner mapping.

Then, the code was executed and it was discovered that when a wall was not detected, the distance was computed from the center to the first point of the image, which caused useless line traces. In order to solve this, it was necessary the use of flags to indicate the presence or absence of walls. Figure 6 shows some further visual errors found.

C. Path planning

We applied the Trémaux algorithm to the obtained maze map. We noticed that some trajectories resulted circular, as shown in Figure 7a. After adjusting the map, we noticed a bigger difference in terms of performance as shown in Figure 7b. It was empirically observed that the resulting routes were more natural and less error prone.

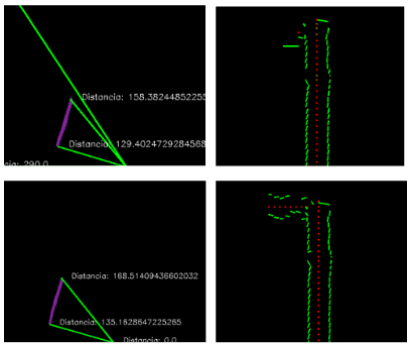
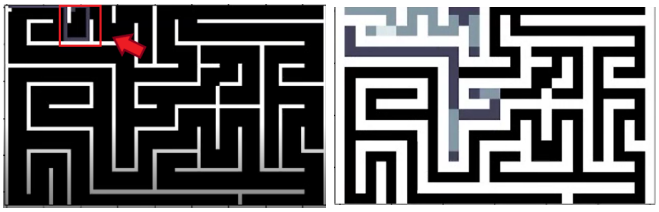


Fig. 6: Wall detection correction.



(a) Trémaux algorithm with the original maze.

(b) Trémaux algorithm with the adjustment maze.

Fig. 7

D. Communication

The main knowledge to be shared between the robots is the map of the maze. This will be used as a guide for the second robot to solve the maze effectively. Currently, the format of the map has not been defined, whether we will communicate it as an image or as a hexadecimal arrangement, it will depend on the mapping module applied.

Notice that the main issue to address in this regard is the Python version incompatibility. On the one hand, Curie will run script in Python 2.7 attending three tasks: (1) acquisition and storage of an image from her upper camera, (2) communication using the JSON standard by Ubidots and (3) its control i.e. the instructions related to the movement of any part of the robot's body. On the other hand, Python 3.7.7 script must be executed in the computer which retrieved the stored images in order to process them. Furthermore, Atom will recover the value of a global variable stored in Ubidots, which contains the positive integer value that represents movement. This variable is updated by Curie depending on the movement to be executed. For instance, if the global variables equal to 8, this translates into rotating the head 90°.

Ubidots is a platform typically used to monitor discrete values, for instance, the temperature of an engine. Nonetheless, it is not a suitable tool for handling large amounts of information, or in our case, pixel and/or hexadecimal chains. Therefore, we looked for platforms that will allow us to do so. Two viable options were found: Firebase and Google Drive. In the red rectangle of Figure 8, it can be seen that a fragment of the hexadecimal chain of a photograph uploaded by Curie in Firebase is enclosed and thus, Atom can make a request

to read this chain. As a result, the robot can perform all the processing with the modules of path planning and mapping.

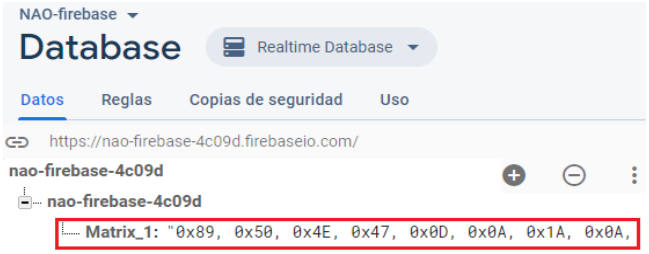


Fig. 8: Firebase matrix.

The Google Drive API was used as follows: from a Python 2.7 script, a Google Sheet was created in which multiple write requests were made to make latency measurements. Figure 9 shows a box plot of the tests carried out. Five tests were implemented, each consisting of sending 23 random character strings with 24 possible combinations with variable length, where the first test had 10,000 characters, the second had 20.000, and so on. When sending the 23^{rd} character string, a console error *Quota exceeded for quota group "WriteGroup"* was shown. In the Google Drive API documentation, it is mentioned that each user can make maximum 100 write requests to a Google Sheet every 100 seconds[13]. Since all the packages were sent in a burst in approximately 27 seconds time, this was the cause of the error.

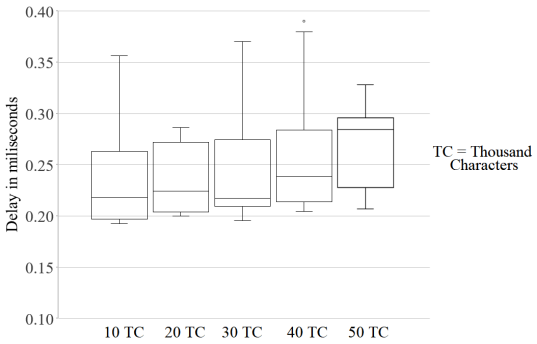


Fig. 9: Write request latency to Google sheets (TC = Thousand Characters).

V. Results

A. Camera calibration

As mentioned in Section IV-A, the ROI of the checkerboard was well defined at that point. The final calibration took approximately 3 minutes to process all images. Although originally designed for webcams, it proved to be effective for the robot camera likewise. Figure 10 shows an example of the result after the 51^{st} iteration.

[13]https://developers.google.com/sheets/api/limits

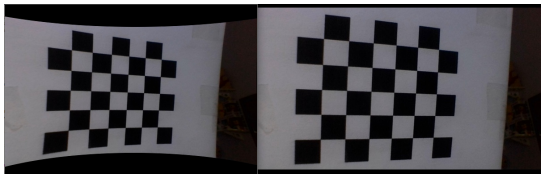


Fig. 10: Image calibration with 51 samples.

B. Mapping

As mentioned in Section IV-B, three fundamental solutions were found to improve mapping. First, wall detection flags were required. Second, we had to stop mapping when an orientation change occurred and restart mapping when both walls were detected again. Finally, in orientation changes, the mapping center must be adjusted. The final result can be observed in Figure 11.

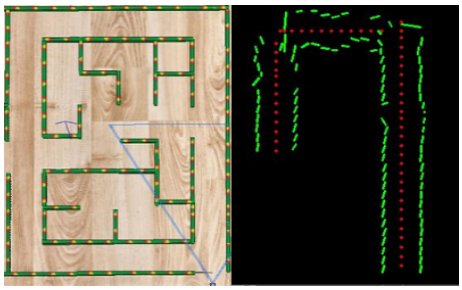


Fig. 11: Sketched maze.

C. Path planning

As concluded from Section IV-C, the Tremaux algorithm is a fast and efficient method to navigate through an unknown maze and find a way out. As seen in Figure 12, despite an initial traverse into a dead end, the robot kept searching effectively until reaching the exit.

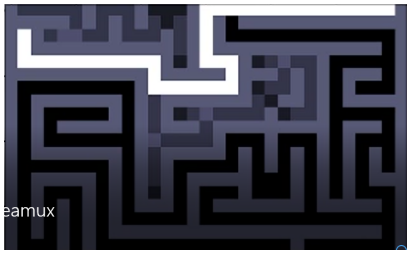


Fig. 12: Tremaux algorithm: path to the endpoint.

D. Communication

Given that Curie and Atom were already connected to Ubidots, a latency test was performed to analyze the time it takes to upload a new value to the global variable in Ubidots and to transfer it into a local variable to be used by Atom. The test consisted of ten different write requests, where the value of the global variable was increased from 1 to 10 in a step of 1. The first execution of the test was performed with

both the computer and Atom connected via Wi-Fi. The results obtained are shown in Table I. We noticed that the average latency obtained from Atom was 42.71 seconds greater than Curie's. To rule out problems with connection speed due to the distance between Atom and the access point, we proceeded to interconnect them via Ethernet. The results obtained were very similar to the reported ones. Again, Atom showed an average latency well above Curie's. It is important to remark that each robot was located in a different place (due to social distancing measures derived from the COVID-19 pandemic), and thus a test within the same network is pending. In addition, it is likely that Atom's network card is damaged due to natural tear and wear.

Test	Curie - sixth version	Atom - fifth version
1	0.6539	43.1109
2	0.6299	43.7829
3	0.5699	43.439
4	0.6779	43.331
5	0.634	43.639
6	0.6189	43.524
7	0.5869	42.78
8	0.6214	43.592
9	0.6385	43.4018
10	0.6452	42.798
Average latency	0.62765	43.33986

TABLE I: Latency (in seconds) to post and get the value of global variable using Ubidots.

From the latency test, it is shown in Figure 13 how Curie (master) raises the left arm and how the number seven is posted in the global variable stored in Ubidots Dashboard. The value of this variable is retrieved by Atom (slave), making the same movement. We provide a video[14] with the complete simulation, where it can be seen that the working principle is the same as in the latency test, but in this case, Atom retrieves a global variable whereas Curie does not.

Fig. 13: Communication between Curie and Atom through Ubidots.

VI. Conclusion

In this paper, we presented a pipeline methodology which allows a robot to solve a maze. Moreover, this robot will be capable of transferring this knowledge onto another robot so that this can solve the maze as well based on different inner and outer settings. Camera calibration could be done correctly using Python 3.7, nevertheless, it needs to be deployed using Python 2.7 to be compatible with the NAO robot. Regarding the mapping step, algorithms used worked correctly when obtaining the maze map. In addition, Tremaux algorithm along with maze adjustment, made possible to obtain the route to get out the maze just by knowing the initial point. Finally, the NAO robots were able to communicate via Ubidots, Firebase, and the Google Drive API, however Atom had higher latency than Curie on the Ubidots platform. The connectivity tests carried out between Atom and the access point lead us to suppose that due to the use of the robot the network connection has flaws. So far, Google Drive API is the most viable platform, since the number of characters we can transmit every 100 seconds is greater than the hexadecimal string that the image represents. Curie could even directly upload the image to a Google Drive folder for Atom to download and process.

In future, other path planning algorithms will be tested in master robot to analyze their speed and effectiveness, intended to reduce the time taken to solve the maze. Also mapping will be improved by determining actual distances or even by reconsidering using corner detection algorithms. Finally, we aim at integrating the whole pipeline into a single system.

[14]https://youtu.be/oDl_gai7iWw

Acknowledgment

The authors would like to acknowledge the financial and technical support of Writing Lab, TecLabs, Tecnologico de Monterrey, Mexico, in the production of this work.

References

[1] X. Cortés, F. Serratosa, and C. F. Moreno-García, "Semi-automatic pose estimation of a fleet of robots with embedded stereoscopic cameras," in *Emerging Technologies and Factory Automation*, 2016.

[2] V. M. Respall, S. Sellami, and I. Afanasyev, *Implementation of autonomous visual detection, tracking and landing for ar.drone 2.0 quadcopter*, 2019.

[3] A. Kumar., P. Biplab., and D. Parhi. (Mar. 2018). Intelligent navigation of humanoids in cluttered environments using regression analysis and genetic algorithm, [Online]. Available: https://doi.org/10.1007/s13369-018-3157-7.

[4] M. Schranz., M. Umlauft., M. Sende., and W. Elmenreich. (2020). Swarm robotic behaviors and current applications, [Online]. Available: file : / / /C : /Users / Areli / Downloads / Swarm_Robotic_Behaviors_and_Current_Applications.pdf.

[5] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[6] J. Matas, C. Galambos, and J. Kittler, "Progressive probabilistic hough transform," 1998.

[7] I. Anureev, "Context machines," in *Proceedings of XVIIIth Concurrency, Specification and Programming*, 2009, pp. 1–12.