Collaborative Robots Assembly Line for Irregular Objects using SLAM

Ayala Oropeza J[1†], Barrón Martínez S[1†], García Pulido J.A[1†], Hernández de la Torre A[1†], Hoyo García J.A[1†], Iglesias Ortiz J.G[1†], Reyna Reyes D[1†], Navarro Guevara R[1†], Rocha Pineda F.E[1†], Trachtman Rivera J[1†]

[1]IT Department, ITESM, Mexico City, 14380, Mexico.

Contributing authors: A01652711@tec.mx; A01652135@tec.mx; A01652094@tec.mx; A01651516@tec.mx; A01658142@tec.mx; A01653261@tec.mx; A01657387@tec.mx; A01652351@tec.mx; A01652082@tec.mx; A01379810@tec.mx;
[†]These authors contributed equally to this work.

Abstract

This paper researches how could three differential mobile robots and two collaborative arm robots cooperate to make an assembly line for objects with irregular shapes. Using algorithms like SLAM, Djekstra path planning path planning

Keywords: Slam, Assembly Line, Collaborative Robot, Differential Robot

1 Introduction

This article presents the use of several robots with the aim of putting them to work together to achieve a specific goal. The project aims to achieve a seamless integration of three differential robots and two manipulator arms, combining advanced robotic technologies with digital systems to optimise industrial processes. Ultimately, it aims to deliver a fully functional and scalable solution that demonstrates the potential of intelligent robotics and the integration of digital systems to solve industrial challenges, resulting in increased efficiency, precision and productivity.

Collaborative robot assembly lines have gained popularity in recent years, for example, companies such as Tesla and BMW [1] have implemented these systems in their

manufacturing processes by using only robots without any or as little human intervention as possible [2]. With this production can increase in efficiency, accuracy and overall productivity. This is happening and it´s getting developed more at increasing speeds because the main aim is that several simple or repetitive tasks that can be very dangerous for a human being can be done by a robot and thus people can focus on more complicated tasks such as planning, designing and validation within the given tasks of an enterprise. Also, by automating these labour-intensive processes, companies can significantly reduce the risk of human error, increase production speed and improve product consistency.

Since the introduction of more complex algorithms and artificial intelligence, robots are now being programmed to be able to do more than just simple, pre-programmed tasks. Companies are looking for them to focus on navigating complex mazes (within the manufacturing plant or big warehouses for example), or to be able to assemble deformed objects and to achieve this with the tools at their disposal that may not be considered the best suited to do the job, but can still be functional. Therefore, new algorithms were implemented such as Simultaneous Localization And Mapping or in other words SLAM which allows the creation of a map of the environment and self-localisation so as not to depend on other more rudimentary algorithms such as odometry and its filters. This eliminates the need for costly and time-consuming retrofitting, making it easier for companies to respond to changing market demands and optimise their production capacities.

2 Delimitation of the problem

In the planning of the project it is expected that two differential mobile robots can carry the lower or upper half of an egg shaped toy respectively, the two half's of the toy have a male and female part that have shapes of vehicles like cars, motorcycles, boats,etc. They also have an unique color in it´s inner part that is shared with it´s specific counterpart. This toy be assembled by applying the necessary force,torque and by making their respective shapes fit in their correct order.

The two differential robots will each carry an specific half part of the toys, being female or male, this with an 3d-printed base that have the capacity of carrying 6 half parts of the egg toy. When the differential robots reach an specified goal by navigating with the Gmapping SLAM algorithm, then they wait in position, this to carry the 3d printed base with the help of an Collaborative arm-shaped robot to a camera that will identify the colors of the eggs with an OpenCV algorithm thus making possible to assemble the toy with its correct counterpart by connecting the camera to the Arm-Robot via ROS. After the robots can assamble correctly the six pairs of eggs these will be delivered to a case reassembling an true egg-shaped box. This box will be then put in a third differential robot that will exit the workspace, finishing the assembly line.

Fig. 1: Image of the toy eggs

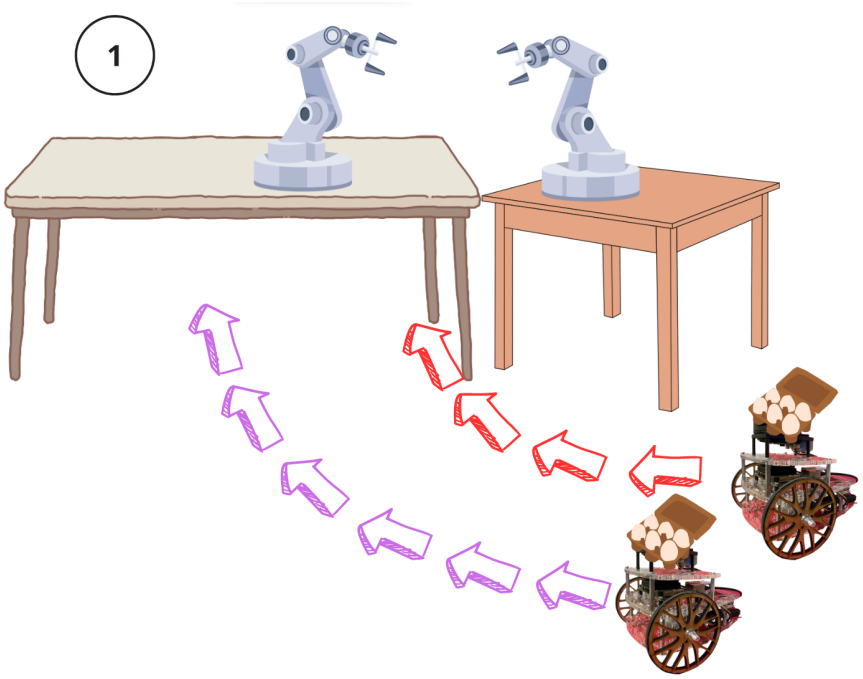


Fig. 2: First part of the estimated functioning of the algorithm

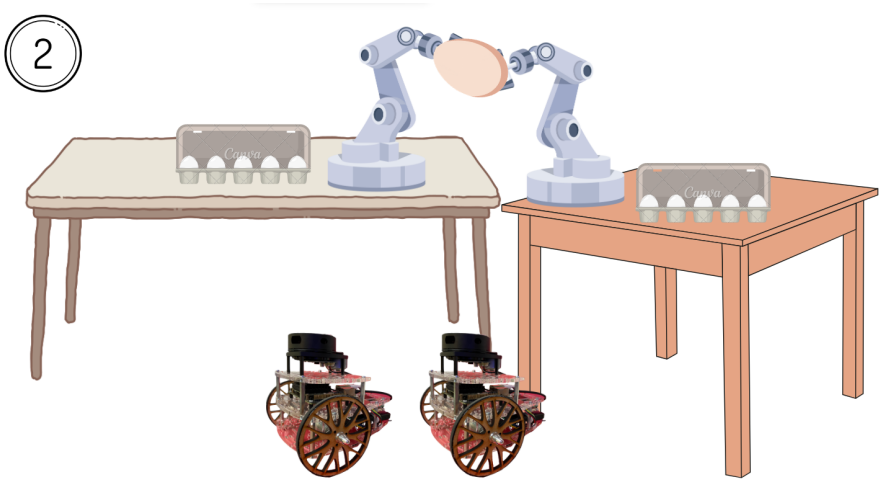


Fig. 3: Second part of the estimated functioning of the algorithm

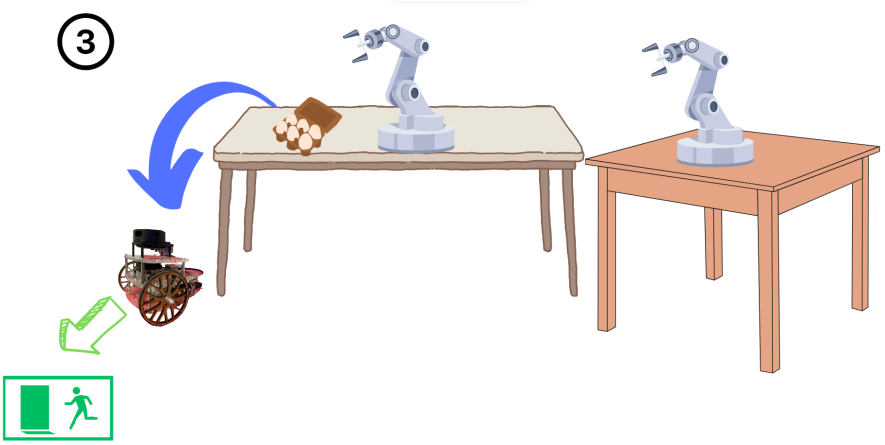


Fig. 4: Final part of the estimated functioning of the algorithm

3 Theoretical Framework

3.1 SLAM

The core technology supporting mobile robots, known as simultaneous localization and mapping (SLAM), plays a crucial role in enabling robots to navigate and understand their surroundings. SLAM involves the integration of real-world sensor data with a robot's internal computer system to create a map of the environment and accurately determine the robot's physical location (Tsubouchi, 2019)[3]. This technology

is widely researched and applied in various domains, such as mathematics, computer implementation, and training, with a primary focus on position estimation and map creation for mobile robots.

SLAM relies on a mathematical framework rooted in probability theory to effectively process and interpret the information obtained from the real-world space. This probabilistic approach allows for precise estimation and representation of the robot's position and the features of its operating environment. The concept of SLAM not only advances the research and development of mobile robots but also enhances understanding in mathematics and proficiency in computer implementation.[4]

3.1.1 GMapping

GMapping, a well-established method in the field of SLAM, utilizes a highly efficient implementation of the Rao-Blackwellized particle filter for learning grid maps from laser range data.

In recent years, the adoption of Rao-Blackwellized particle filters has emerged as an effective solution for addressing the SLAM problem. This approach leverages a particle filter framework where each particle represents an individual map of the environment. However, a critical concern in this context is the need to minimize the number of particles to enhance computational efficiency. To tackle this challenge, adaptive techniques are introduced to reduce the number of particles in a Rao-Blackwellized particle filter designed for learning grid maps. An innovative approach is proposed to compute an accurate proposal distribution that considers not only the robot's movement but also the most recent observations. By incorporating this information, the uncertainty pertaining to the robot's pose during the prediction step of the filter is significantly reduced (OpenSLAM, 2018)[5].

Occupancy grid mapping is a widely employed technique in robotics and SLAM, utilizing a grid-based representation to discretize the continuous environment. Sensor measurements are used to calculate the likelihood of occupancy for each grid cell, leveraging a probabilistic framework (Souza, 2012)[6]. This uncertainty is represented by maintaining a probability value for each cell, indicating the likelihood of it being occupied or unoccupied. The grid is refined and accuracy improves as the robot explores and collects more sensor data. The resulting occupancy grid serves various purposes, including navigation, path planning, and obstacle avoidance. It offers a structured representation of the environment, allowing the robot to make informed decisions based on the occupancy information. By converging probabilities of occupied or unoccupied cells, a reliable representation of obstacles is obtained. Using a grid-based mapping algorithm, is instrumental in creating a precise map of the laboratory or its surrounding environment for a puzzlebot. As the puzzlebot explores the environment, it updates the occupancy probabilities based on sensor measurements obtained from Lidar and camera (Souza, 2012)[6]. By utilizing G mapping, the puzzlebot constructs an accurate and current map of the laboratory and its surroundings. This map serves multiple purposes, including path planning for navigating around the maze and identifying the next locations. The acquired map empowers the puzzlebot to make informed decisions and execute puzzle-solving tasks efficiently, such as grasping and relocating the basket to their appropriate positions.

3.1.2 LIDAR

LIDAR sensors provide several advantages in the realm of SLAM and mapping: Advantages: LIDAR sensors deliver highly accurate and dense 3D point cloud data, enabling precise mapping and localization. They exhibit superior resilience to varying lighting conditions, surpassing the capabilities of cameras. This attribute allows LIDAR sensors to operate reliably in both indoor and outdoor environments. Additionally, LIDAR sensors offer a wide field of view, capturing significant spatial information.

However, it is important to consider the limitations of LIDAR sensors: LIDAR sensors tend to be relatively expensive and physically bulky, limiting their deployment on compact or lightweight robotic platforms. Transparent or reflective surfaces can pose challenges to LIDAR-based mapping and localization, as they may result in incomplete or distorted data. Moreover, occlusions caused by objects in the environment can also hinder the performance of LIDAR-based systems. Given their characteristics, LIDAR sensors are particularly well-suited for tasks that demand high-precision mapping, 3D reconstruction, and obstacle detection. They find prominent applications in domains such as autonomous driving, robotics navigation, and industrial settings where robust and accurate spatial awareness is crucial.

3.2 Path planning

Path planning plays a crucial role in autonomous robotic systems, allowing them to navigate efficiently and safely in complex environments. Various navigation algorithms have been developed to address this challenge, including Dijkstra's algorithm, potential fields, wall follower, and line follower. These methods offer different approaches to finding optimal paths and avoiding obstacles.[7]

Dijkstra's algorithm is a well-known graph-search algorithm widely used for path planning. It explores the graph representing the environment, considering the costs associated with each edge, and determines the shortest path from a start point to a goal point. This algorithm guarantees an optimal solution but can be computationally expensive in large-scale environments.[8]

Potential fields, on the other hand, are based on the concept of simulating forces in the environment. The robot perceives attractive forces towards the goal and repulsive forces from obstacles. By balancing these forces, the robot can navigate towards the goal while avoiding collisions. Potential fields are computationally efficient but may suffer from getting trapped in local minima or facing difficulties in handling complex environments.[9]

The wall follower algorithm is commonly used for navigating along the boundaries of an environment. The robot follows the walls, maintaining a constant distance from them. This method is effective in simple maze-like environments but may encounter challenges in more complex layouts or open spaces.

The line follower algorithm relies on detecting and following lines or paths on the ground. Sensors such as cameras or infrared sensors are commonly used to track the lines and guide the robot's movement. Line followers are often employed in applications like automated guided vehicles (AGVs) or line-following robots.

These path planning and navigation methods provide different trade-offs in terms of optimally, computational complexity, and suitability for specific environments. We have studied and applied these techniques in to our robotic system puzzlebot.

3.3 Manipulators UR5

The Universal Robots UR5 is a popular and versatile collaborative robot manipulator widely used in various industrial applications. With its compact design, high payload capacity, and intuitive programming interface, the UR5 offers flexibility and ease of use for automation tasks.[10]

The UR5 is equipped with six degrees of freedom, allowing it to perform complex and precise movements. It has a payload capacity of up to 5 kilograms, making it suitable for a wide range of assembly, pick-and-place, and machine tending tasks. The robot's lightweight design and built-in safety features enable it to work alongside human operators without the need for safety barriers.[11]

One of the key advantages of the UR5 is its easy programming interface. The robot uses a graphical user interface (GUI) called Polyscope, which allows users to program and control the robot intuitively. Polyscope offers a range of programming options, including teach pendant programming, drag-and-drop functionality, and the ability to integrate with external sensors and systems.

The UR5 also supports various communication protocols, such as Modbus, TCP/IP, and Ethernet/IP, enabling seamless integration with other automation equipment and systems. This facilitates the coordination and synchronization of multiple robots or the interaction between the UR5 and external devices.

3.4 UR cap ROS

URCap ROS (Robot Operating System) is a powerful tool that enables seamless integration of Universal Robots (UR) with the ROS ecosystem. This integration opens up a wide range of possibilities for advanced robot control, perception, and collaboration in various applications.

The URCap allows for the calling to ROS services, actions and topics within the UR operating system configuration from the Teach tablet (Polyscope). The main program is installed on the robot via an external USB with the extension installation. Once installed on the robot, a server-like tcp connection has to be launched on the computer in order for the robot can also connect and thus establish a message communication between the robot and the computer. [12]

3.5 Open CV

OpenCV (Open Source Computer Vision Library) and NumPy are two widely used and powerful libraries in the field of computer vision and image processing. Together, they provide a comprehensive set of functions and tools for working with images, including color identification.

OpenCV is an open-source library that offers a wide range of image processing functions and algorithms. It provides a convenient interface for loading, manipulating, and analyzing images. OpenCV supports various programming languages such as Python, C++, and Java, making it accessible to a broad community of developers.

NumPy, on the other hand, is a fundamental library for scientific computing in Python. It provides a powerful multidimensional array object and a collection of functions for performing array operations efficiently. NumPy's array structure allows for fast and efficient computation, making it ideal for working with image data.

4 Methodology

4.1 Puzzlebot operations

4.1.1 Robot's description

As explained in Section 2 the main objective for our mobile robots was to deliver the egg shaped toys to the robotic arms. We used two Puzzlebot Jetson Edition to achieve this task. The two robots we used are called "Hernan" and "Frank-bot". Both of this robots were equipped with a Rplidar A1M8-R6 with the characteristics specified in the Table 1, a NVIDIA Jetson Nano [13] and a Raspberry Pi Camera PIS-1685 described in Table 2.

After the assembly has been completed, we use a third robot to take the egg shaped toys out of the lab, this third robot's name is "Motomami". This robot is equipped with a Rplidar A1M8-R6 with the characteristics specified in the Table 1 and a NVIDIA Jetson Nano.

Parameter	Value	Units
Range	12	m
Sampling frequency	5.5	Hz
Angle increase	0.004	rad
Distance resolution	0.2	cm

Table 1: Characteristics of the Rplidar A1M8-R6 [14]

Parameter	Value	Units
Resolution	1080	pixels
FOV	160	degrees
Focal length	3.15	mm
Distortion	14.3	%

Table 2: Characteristics of the Raspberry Pi Camera PIS-1685 [15]

4.1.2 Environment

To improve the localization estimation, we created a closed space inside our workspace, as we can see in Fig.5 and Fig.6. After creating the environment we created a map using SLAM which created the result we can see in Fig. 7.

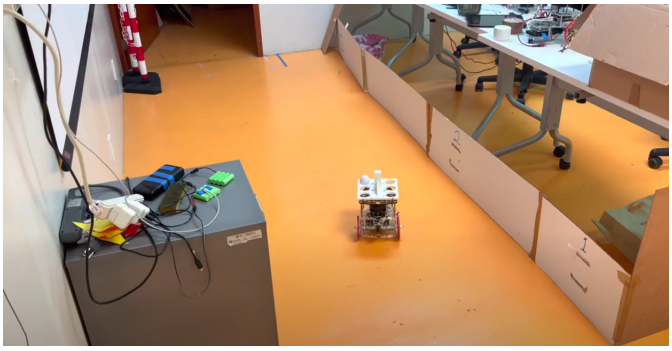


Fig. 5: Photo of the beginning of the environment

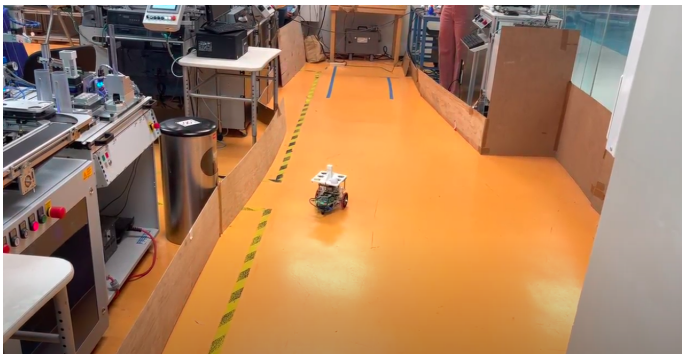


Fig. 6: Photo of the end of the environment

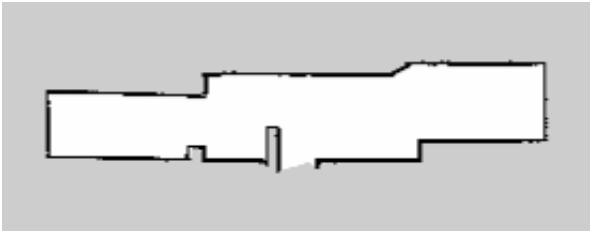


Fig. 7: Map created using SLAM

After creating the map, we were able to define the starting and final goal for both delivering robots. The positions are defined in Tables 3 and 4; and shown in Fig. 8. The end positions were defined based on the robotic arm's area of reach.

	Start position	End position	Units
x	0.3	10.6	m
y	1.1	1.9	m
yaw	0.0	0.0	rad

Table 3: Start and end positions for Hernan.

	Start position	End position	Units
x	0.3	9.3	m
y	0.6	1.1	m
yaw	0.0	0.0	rad

Table 4: Start and end positions for Frank-bot.

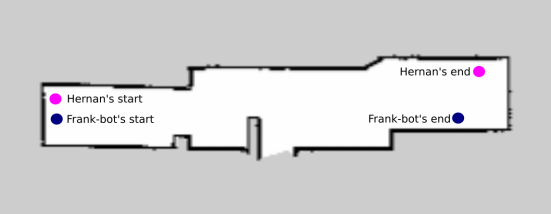


Fig. 8: Start and end positions for the delivering robots.

We did the same process for our exiting robot, whose start and end points are defined in the Table 5 and in the Fig. 9.

	Start position	End position	Units
x	10.8	4.2	m
y	0.8	0.0	m
yaw	π	$\frac{\pi}{2}$	rad

Table 5: Start and end positions for Motomami.

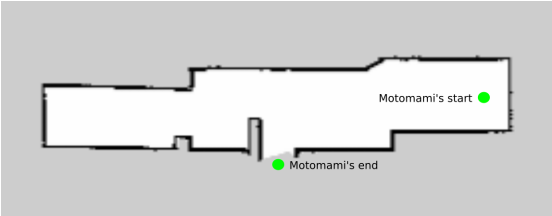


Fig. 9: Start and end positions for the exiting robots

4.1.3 Odometry

In order to create an initial estimate of our robot's position, we used the Kinematic model of a differential robot to obtain it's current speed, the model for the linear speed of the robot can be seen in (1) and the model for the angular speed can be seen in (2)

$$v = r\frac{\omega_r + \omega_l}{2} \tag{1}$$

$$\omega = r\frac{\omega_r - \omega_l}{l} \tag{2}$$

Where:
$v \rightarrow$ Linear speed
$\omega \rightarrow$ Angular speed
$\omega_l \rightarrow$ Angular speed for the left wheel
$\omega_r \rightarrow$ Angular speed for the right wheel
$r \rightarrow$ Radius of the wheels
$l \rightarrow$ Distance between the wheels

After obtaining the speed of our robot, we used Dead Reckoning to obtain an estimate of our robot's position using (3), (4) and (5)

$$\theta[k] = \theta[k-1] + \omega[k]dt \tag{3}$$

$$x[k] = x[k-1] + v[k]cos(\theta[k])dt \tag{4}$$

$$y[k] = y[k-1] + v[k]sin(\theta[k])dt \tag{5}$$

We implemented this using rospy following the flowchart in Fig. 10 and updating it with a frequency of 10 Hz.

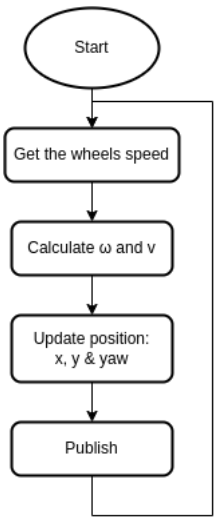


Fig. 10: Flowchart for the odometry calculation algorithm

4.1.4 Tf publisher

To update the position of our robot's sensors relative to the chasis, we need to update a transform tree. Our robot consists of multiple links shown in Table 6, with their transformation matrices shown at the initial position, this links and their joints are defined using a .xacro file that's used to create an URDF.

Name	Abbreviation	Type of joint	Parent link	tf matrix
base_footprint	bf	None	None	None
base_link	bl	Static	base_footprint	(6)
chassis	ch	Static	base_link	(7)
camera	ca	Static	chassis	(8)
laser	la	Static	chassis	(9)
left_wheel	lw	Revolute	chassis	(10)
right_wheel	rw	Revolute	chassis	(11)

Table 6: Links of our robot

$$ {}^{bf}T_{bl} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.05 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6} $$

$$ {}^{bl}T_{ch} = \begin{bmatrix} 1 & 0 & 0 & -0.05 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7} $$

$$ {}^{ch}T_{ca} = \begin{bmatrix} 0.97 & 0 & 0.24 & 0.09 \\ 0 & 1 & 0 & 0 \\ -0.24 & 0 & 0.97 & 0.05 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8} $$

$$ {}^{ch}T_{la} = \begin{bmatrix} -1 & 0 & 0 & 0.06 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0.1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9} $$

$$ {}^{ch}T_{lw} = \begin{bmatrix} 1 & 0 & 0 & 0.05 \\ 0 & 1 & 0 & 0.09 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10} $$

$$ {}^{ch}T_{rw} = \begin{bmatrix} 1 & 0 & 0 & 0.05 \\ 0 & 1 & 0 & -0.09 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11} $$

To publish and update the transforms of the links defined in the .xacro file we use the packages robot_state_publisher and joint_state_publisher. But, since the transform between base_footprint and base_link is not included inside the definition provided by the robot's manufacturer, we publish it using our own code.

This code also publishes a transform between the odom frame and the base_footprint this transform is defined by the dead reckoning published by the odoemtry. This code works following the flowchart in Fig. 11 and updating it with a frequency of 10 Hz.

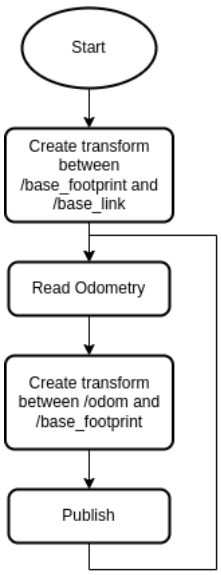


Fig. 11: Flowchart for the transform publisher algorithm

4.1.5 Path planning

As part of our navigation, our robot uses the map generated by Gmapping. We use this map and the Dijikstra algorithm to create a path. For this path we consider the unkonwn positions as occupied space. Because of this, the path planning algorithm will not create a path, in this scenario we find the empty space closest to the objective and navigate towards that.

This algorithm follows the flowchart in Fig. 13 and updating it with a frequency of 10 Hz.

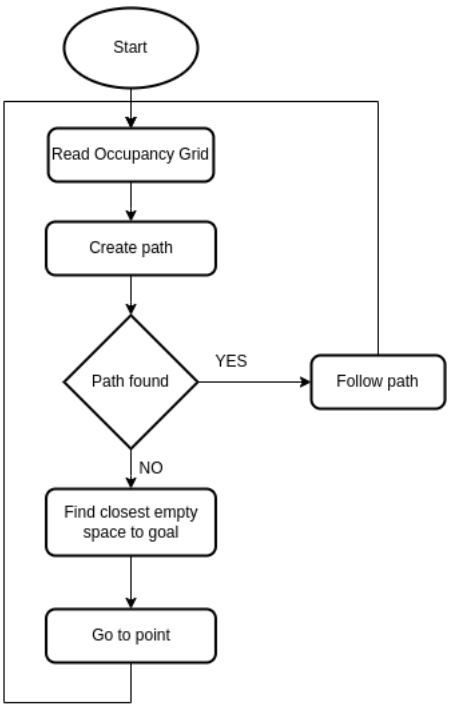


Fig. 12: Flowchart for the path planning algorithm

4.1.6 Navigation: Potential fields

After publishing our path or our objective, we navigate using potential fields. For this algorithm we create a repulsion force taking into account the obstacles that are within a 0.5m radius circle. Then we represent each of them using a vector with it's magnitude being inversely proportional to their distance to the robot. Then we do a similar process whit the objective point, but this point is used to create an attraction force. To obtain the total force applied to the robot, we add every force calculated to obtain a final vector. The combination between a path planner and a the reactive navigation allowed us to find the optimal route to an objective while still being able to react to dynamic obstacles.

Once we have the final vector, we define our linear and angular speed, as we can see in the flowchart in Fig. 13 and we ran this algorithm with a frequency of 10 Hz.

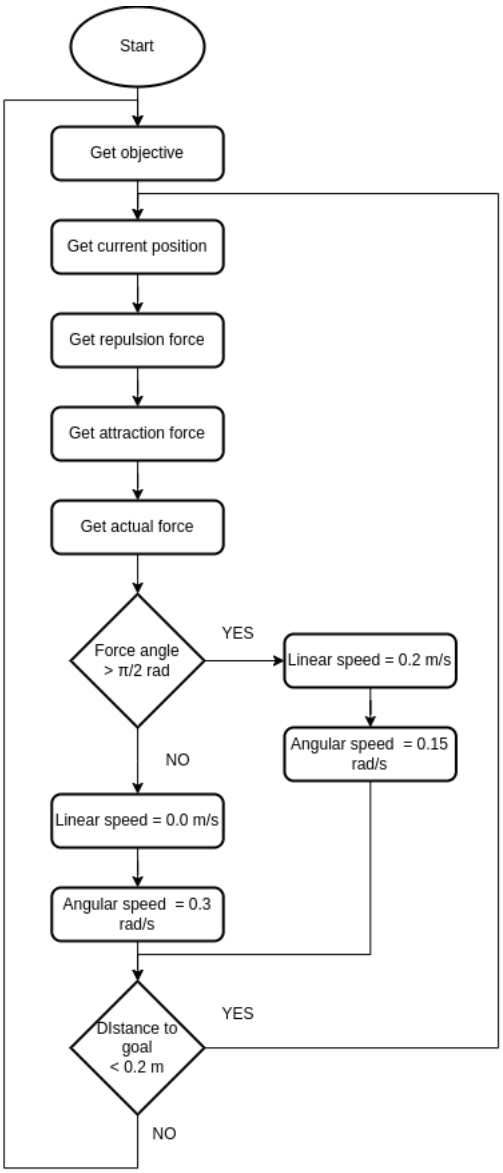


Fig. 13: Flowchart for the potential fields navigation algorithm

4.1.7 Navigation: Wall follower

One way we can reduce the uncertainty of our position is to follow a wall with a defined separation. In this scenario we follow the wall to the left of the robot, with Hernan using a limit of 0.6 m and Frank-bot using a limit of 1.2 m.

This algorithm was implemented using a state machine, where we have a state that rotates the robot until there's a wall to it's left at the defined limit. Another state consists of moving in a straight line and the final state combines both state so the robot rotates and moves forward. The state machine diagram can be seen in the Fig. 14.

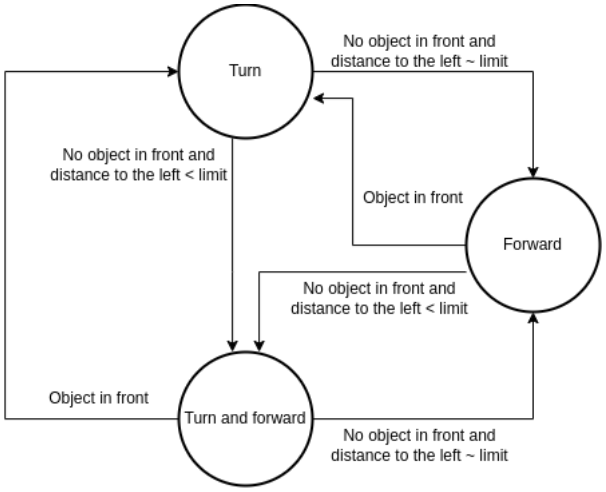


Fig. 14: State machine for the wall follower algorithm

4.1.8 Navigation: Line follower

Since we needed to guarantee a high precision for our robotic arm to grab the toys, we used a line follower to help us increase the precision. For this algorithm we used a blue line, given that the floor of our environment was orange and that these colours are on opposite sites of the colour wheel.

For this algorithm we used the Raspberry Pi Camera PIS-1685 of our two delivering robots, but, in order to improve the speed of the algorithm, we only use the lower sixth of the image with reduced definition. Once we have the cropped image, we find the center of mass of the blue object of the image, and the algorithms goal is to keep this center of mass in the center of the image.

This algorithm follows the flowchart in Fig. 15 and updating it with a frequency of 10 Hz.

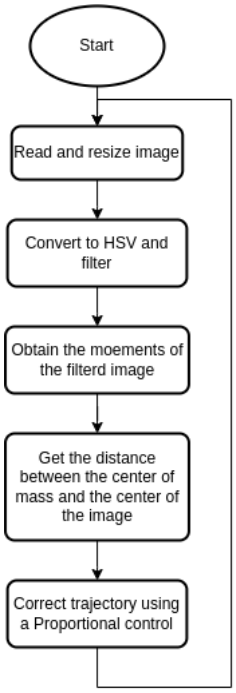


Fig. 15: State machine for the line follower algorithm

4.1.9 Pilot

Having the navigation algorithms defined before, we created an algorithm that decides which one of the behaviors to use. This decision was based primarily in the robot's position, since, after much testing, we found out that navigation using path planning and potential fields induced uncertainty, aggravated by the fact that we were mapping while delivering the egg shaped toys. Because of this, we used the wall follower algorithm to align the robots with the walls, and, after that we follow the line until we reach the objective. Once that's happened, we send a Boolean message to our robots so they can start their routine.

The workings of this final algorithm can be seen in the flowchart in Fig. 16 and updating it with a frequency of 10 Hz and it's used by the two delivering robots. The exiting robot only uses the path planning and Potential fields algorithms defined before.

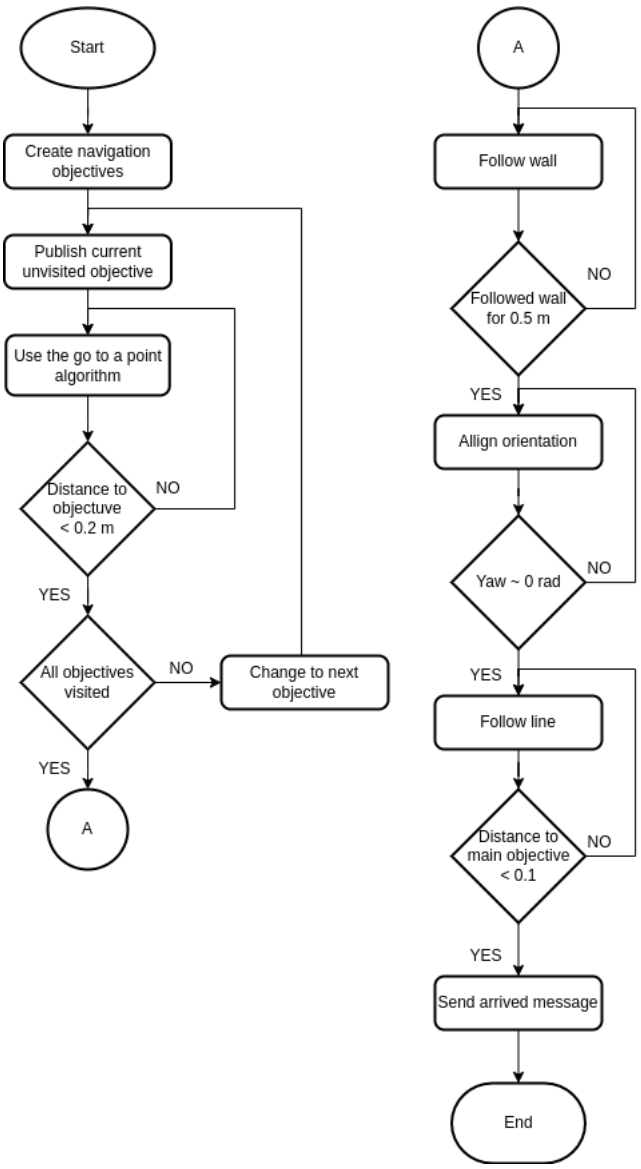


Fig. 16: State machine for the pilot algorithm

4.1.10 View using rqt_graph

Everything we talked in this section was implemented in different nodes, alongside Gmapping SLAM. The connections between the different nodes can be found in the Fig. 17.

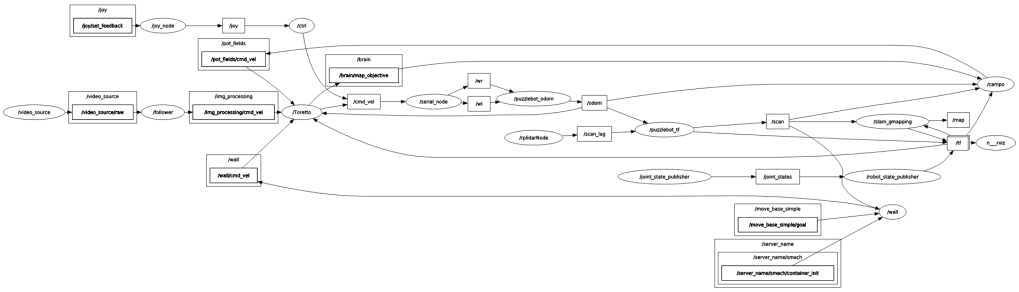


Fig. 17: Rqt_graph for one robot

4.2 UR5 routines

When the puzzlebot robot arrives at the unloading area, the UR arm picks up each tray (one being classified as male and another one as female) and passes it over a camera to detect the colors of each of the eggs. This is used to know the order in which they were placed in the tray. Once the order is obtained, both trays are positioned on the table so that both UR arms grab the eggs in the order detected and assemble them in their color. Once each egg is assembled they are stored in the final tray where the other eggs are located. Finally when all the eggs are put on the final tray, one UR grabs the tray and puts it on another puzzlebot and takes it back to the starting point. The main routine is divided in three subroutines:

1. Pick up the tray and show it to the camera.
2. Join the halves of the eggs and put them in another tray.
3. Close the tray and put it on the last puzzlebot.

Figure 18 shows the flowchart of this routines.

4.2.1 Pick up the tray and show it to the camera.

The first routine waits until the signal on the respective ROS topic indicates that the puzzlebot has arrived to the station. Then moves the arm to the tray, grabs it and shows it to the camera and wait for another signal on the ROS topic to know when all the eggs are succefully detected. Finally, leaves the tray on the table.

4.2.2 Join the halves of the eggs and put them in another tray.

The second routine grabs each of the egg halves and moves it to the assemble position. Once there, the arm waits for a signal of the other robotic arm to know that both are ready and in their assemble positions, then, both move closer. Only one arm rotate and will be checking the torque and the force in order to know when the egg is fully joined. Finally, one robot release the egg and the oder one puts it on the final tray.

4.2.3 Close the tray and put it on the last puzzlebot.

The last routine closes the final tray and puts it on the other puzzlebot. After that, it sends a signal to the ROS topic for the puzzlebot to know that it can leave.

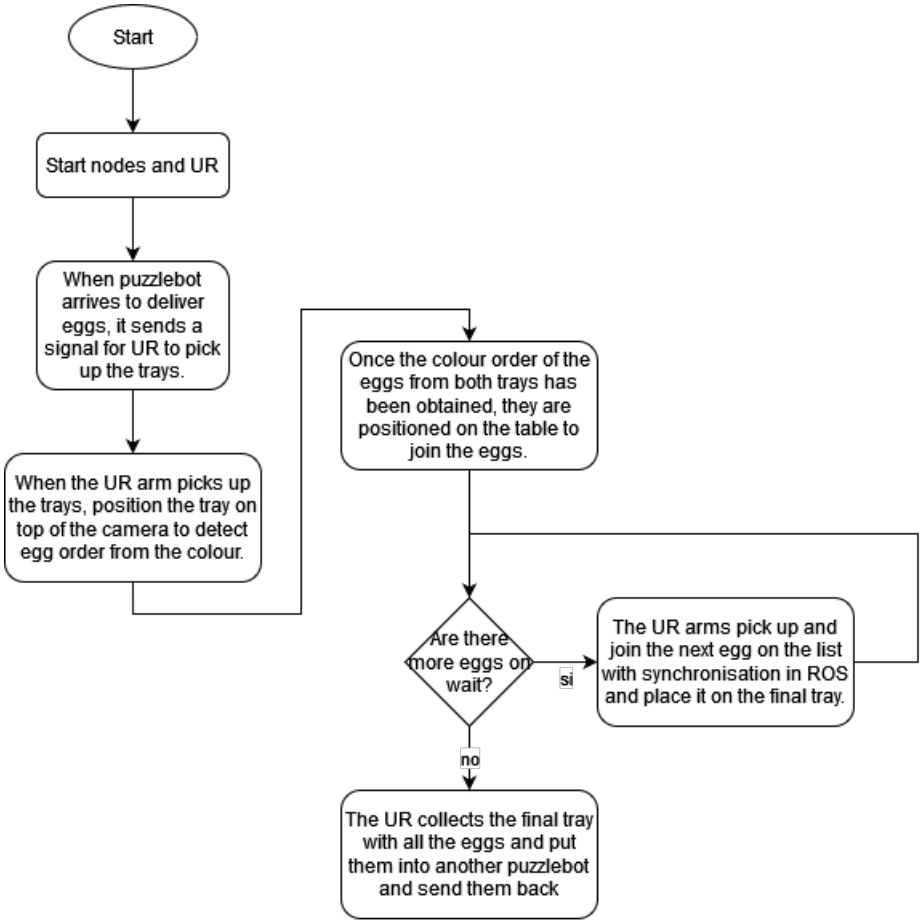


Fig. 18: FLowchart of the UR5 routines

4.3 Color identification

For image processing, the OpenCV and NumPy libraries were used. We utilized a webcam with a resolution of 1920x1080 to capture the images. In conjunction with the RosBridge plugin in the robotic arms, a connection was established between the ROS nodes generated for color processing and sending the location of the identified color to the manipulator arms.

First, the image was captured from the camera using OpenCV. Then, the RGB (red, green, blue) values of the image were converted to HSV (hue, saturation, value) values. This allows us to create HSV ranges for detecting different colors. To achieve this, it is necessary to define a lower and upper limit where the color to be detected may be found. The respective values for each color are presented in the following table 7. With the obtained HSV values for each color, the next step is to store the twelve

Table 7: Values for each color

Color	Lower limit	Upper limit
Red	170, 140, 0	195, 195, 255
Yellow	13, 30, 0	55, 95, 255
Purple	110, 50, 0	135, 125, 255
Orange	0, 80, 0	75, 156, 255
Blue	100, 150, 0	110, 215, 255
Green	60, 40, 0	99, 165, 255

ranges in a list so that the color masks can be changed based on a flag indicating the possibility of mask switching. This enables color detection to be modified without manual intervention. Subsequently, a 2x2 matrix (kernel) is created to clean the image. This involves dilating the pixels that detect the desired color and then eroding the pixels to eliminate as much noise as possible.

Once the color mask without noise is obtained, the camera image is divided into six parts. Each part focuses on one of the six areas of the base where the objects to be manipulated will be placed. This ensures minimal processing for expedited color detection. Finally, color densities are calculated to determine the amount of color observed by the camera at any given time. The formula used for density calculation is as follows:

$$\rho = \sum_{x,y=0}^{k,L} \frac{v[x][y]}{L * 255} \tag{12}$$

Where x and y represent the dimensions of the image (width and height, respectively), and L is the size of the image. This allows us to ascertain the number of pixels detecting the desired color. Subsequently, the cropped images with the applied color mask, as well as the densities of each image, are published. These density messages are utilized in another file to identify the box with the highest density. This is achieved by storing all densities in a list and using the np.max function. When calculating the highest density, it is verified that the calculated value is greater than 0.2. This ensures that the object is located in the designated area. Once these steps are completed, the index of the highest density is obtained and published. This index corresponds to the position of the object on the base. This value is then sent to the robotic arm, which has the coordinates of each position stored, and is responsible for grasping and manipulating the object.

Flowchart look figure 19

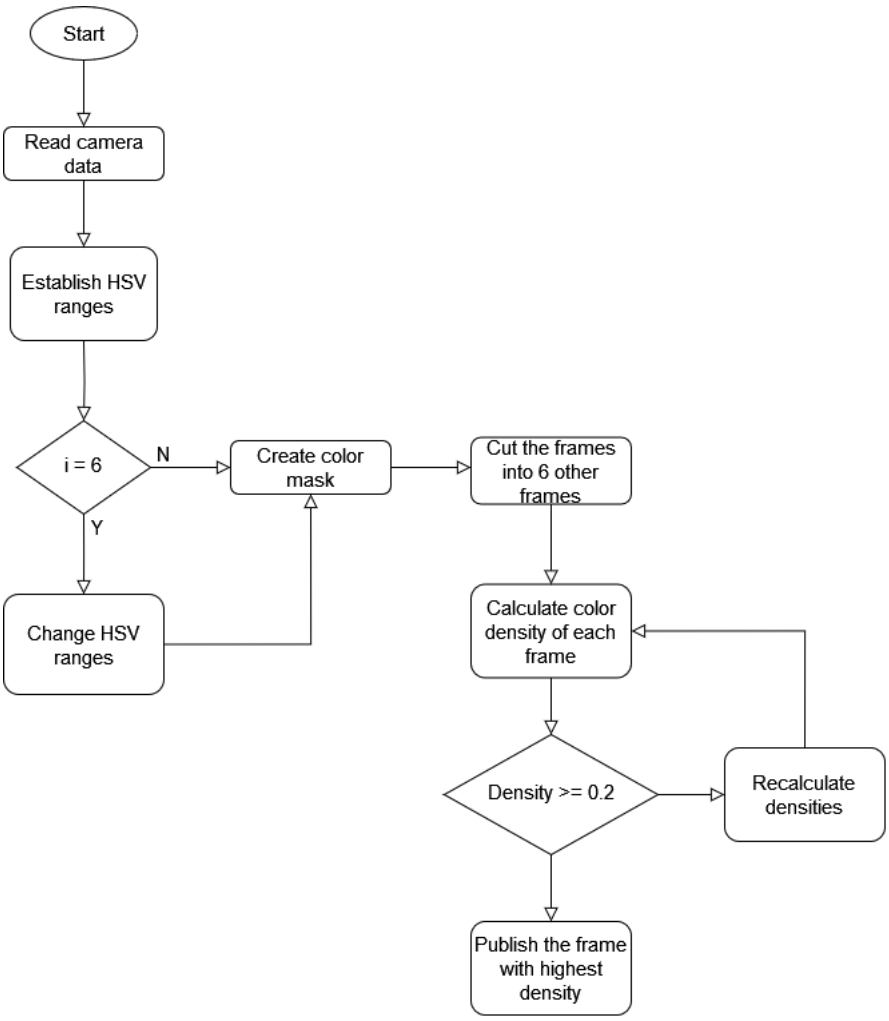


Fig. 19: Flowchart image processing

4.4 Collaboration with the robots

In order for our robots to collaborate we created a LAN using a class C private IP using a TP-LINK Archer 7 Router. The main objective was for all the robots to communicate to the same roscore so they can share messages.

4.4.1 Devices

The devices we used for our collaboration alongside their description and the static IP assigned to them can be found in Table 8.

Device name	Description	IP
puzzlebot-master	Creates all the nodes needed for navigation and mapping for the three mobile robots while hosting the roscore for the whole assembly line	192.168.0.117/24
hernan	Mobile delivering robot	192.168.0.114/24
Frank-bot	Mobile delivering robot	192.168.0.120/24
Motomami	Mobile exiting robot	192.168.0.123/24
UR-master	Controls the communication with the UR5 robots and the color identification using the camera	192.168.0.129/24
UR5-1	Robotic assembly arm	192.168.0.141/24
UR5-2	Robotic assembly arm	192.168.0.161/24

Table 8: Collaborative devices

4.4.2 Network connections

The devices are connected to the network using the connection diagram in Fig. 20.

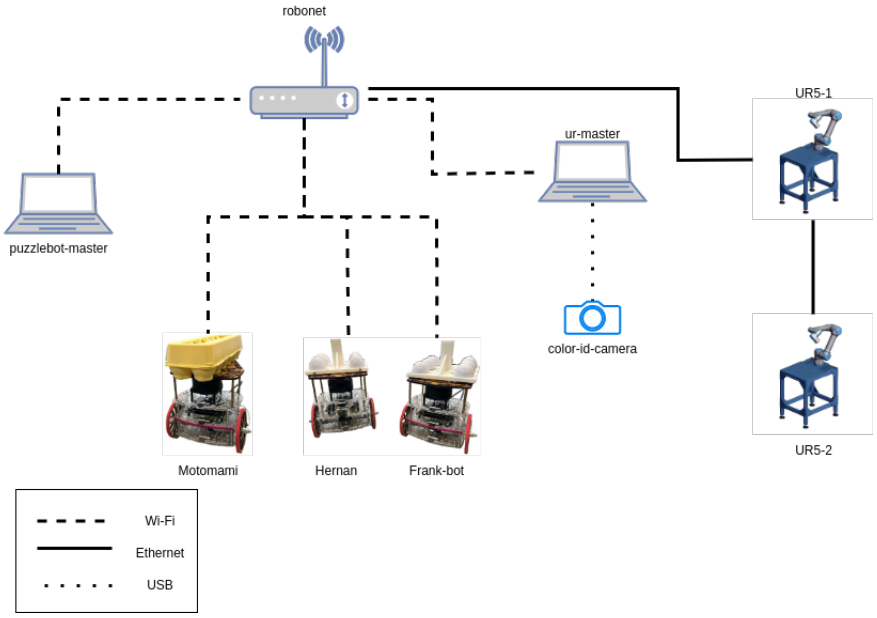


Fig. 20: Netwrok connection diagram

4.4.3 Collaboration using ROS

In order to communicate between different robots that use the same topics, such as /cmd_vel for the differential robots, we created namespaces for each robot. Each robots namespace is defined in Table 9.

Robot	Namespace
Hernan	/he
Frank-bot	/fb
Motomami	/mm
UR5-1	/up
UR5-2	/up

Table 9: Namespace for the collaborative devices

5 Results

5.1 Image processing

As for the computer vision part, both color processing and object position detection were successful. The results of the image detection algorithm can be viewed in the following table of images.

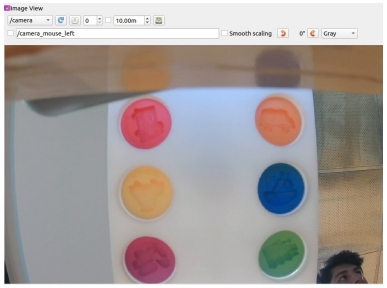
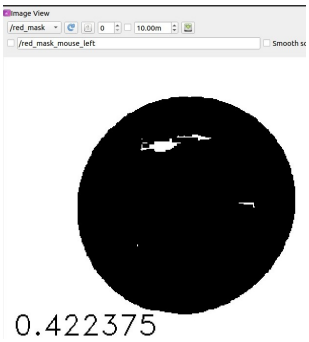
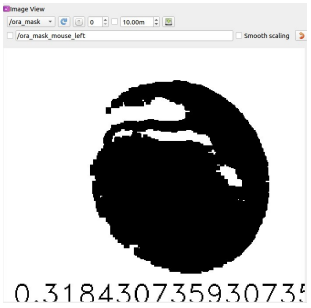


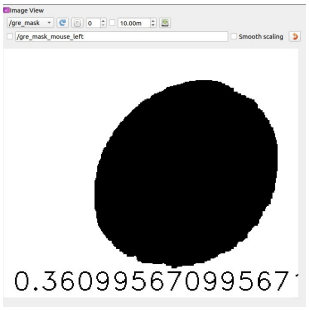
Image viewed from the camera



Color density of red



Color density of orange



Color density of green

Table 10: Images Viewed from the Camera and Color Detection

As can be seen in the black and white figures from Table 10, the density of the detected color could oscillate between 30% and 45%, while in the worst cases, the density could go even lower, to a 20% of density.

The algorithm was able to perceive the desired color 80% of the time, while the other 20% that was not detected depended on several factors, one of them being the illumination of the place where the tests were carried out. This factor alone, combined with certain combinations of the position of the objects in the trays, could cause the detection algorithm to be able to pick up some color in the image, but the detection was not good enough to pass the threshold established to declare that color was being detected in a given location. In order to solve the above problem, another algorithm could be generated that would essentially have the same detection capabilities but taking into consideration the time of day at which the test is being performed, thus having a dynamic way to change the color masks to work at any time of the day and having the same effectiveness.

On the other hand, the detection of the position of the object and the sending of this information to the robotic arms was also effective, however, when trying to send the information from ROS to the operating system of the arms (Polyscope) sometimes it was not received in the latter. This may be due to the fact that the plugin used in Polyscope is still in the early stages of development and latency or synchronization issues may be caused by this aspect. Despite the above, each time the data was generated and sent successfully, the operating system was able to receive the data favorably. So, in order to solve this problem, the best solution is to wait for the developers to generate newer versions of this software so that the robotic arms are now able to receive any type of information in a frequent and effective manner.

5.2 SLAM and path planning

As dictated by our navigation pilot, our robot begins navigation using potential fields and path planning. But, when a path is created it doesn't create an optimal path. As we can see in Fig. 21, but, since we consider a point has been visited when a robot's is within 0.2 m, the trajectory the robot follows is closer to an optimal route.

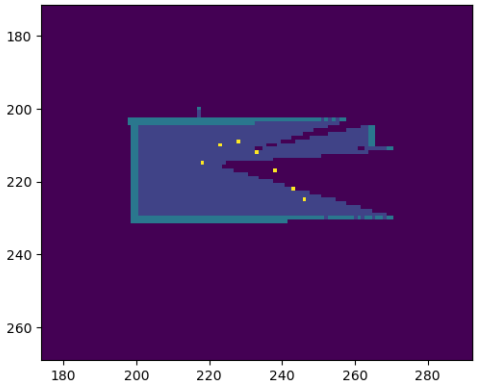


Fig. 21: Example of a path created

Both delivery robots follow the route until the beginning of the line (Fig. 23) while avoiding obstacles (Fig. 22).

Fig. 22: Hernan avoiding an obstacle

Fig. 23: Hernan at the beginning of the line

After following the wall, our robot begins following the line (Fig. 24) until it arrives to the final position (Fig. 25).

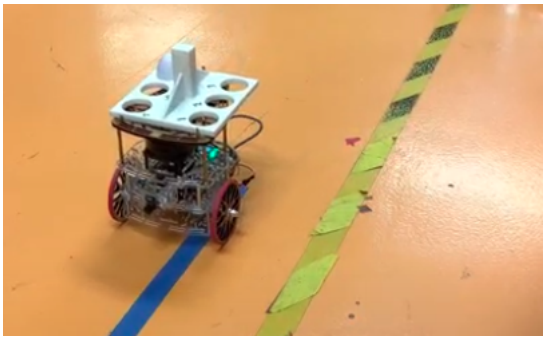


Fig. 24: Hernan following the line

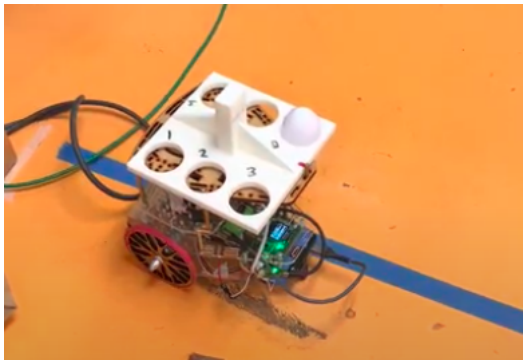


Fig. 25: Hernan at the end position

During this process, the robot is mapping as shown in Fig. 26. This process is followed by both delivering robots, while the exiting robot maps the environment (Fig. 27) and exits the lab (Fig. 28).

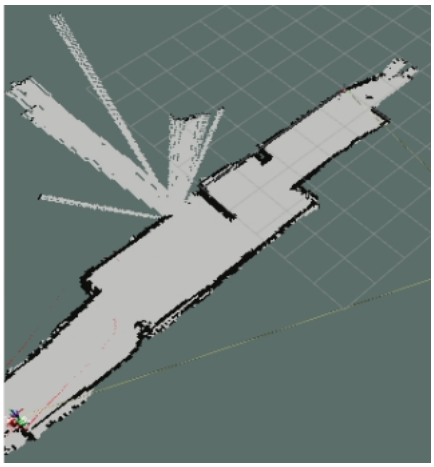


Fig. 26: Map created by Frank-bot

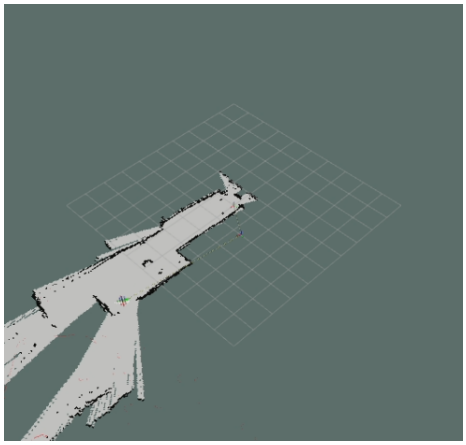


Fig. 27: Map created by the exiting robot

Fig. 28: Exiting robot exiting the environment

After running this test 10 times, we determined that 60% of the test were successful, while the others we're affected by environment factors such as the lighting or the precision of the corrections created by SLAM.

Of those test that were successful, we found that the final position has an error of $+/-0.05m$ and $+/-5^o$, and, while this error is within the expected range of the localization of a mobile robot, it's higher than the expected error so there can be a flawless interaction between the mobile robots and the robotic arms.

Even though different methods were tried, such as rails to align the robot, we did not manage to reduce the error, making it so that our interaction between the robots requires a human intervention. But, this could be improved if we had the map before hand and navigate and locate the robot's position using an algorithm such as amcl.

5.3 UR5 assembling

5.3.1 Grasping of the base and its positioning

For the assembling of the eggs we first needed to grab the 3D-printed base where the half's eggs where located, in order to do this we started estimating the end location of the differential robot specifically Hernan. The error that Hernan had at the endpoint, also affected how we grabbed the base and also gave us an error that was crucial in the pose where the arm had to be . We tried the experiment approximately 20 times and 13 times the position of the base was inconsistent, but in an acceptable range to the program to function correctly, this affected the experiment negatively and thus in the correct grasping of the egg.

Fig. 29: Arm with the base placed in it´s correct position

5.3.2 Grasping of the egg

To solve this problem we use the function included in the programming of the UR5 system. This function allows the robot tool to go down in the estimated position of the egg depending on where we left the base, if our position was correct the robot would detect and collision and retract, after that the robot would grab the egg. in this part we had two grippers to grab the egg. the first one from the company robot On didn't´t have any problem in grasping, contrary to this the gripper from the company Robotiq struggled because of the slippery nature of the toy and its shape. To solve this issue different methods were tried with varying degrees of success. In those alternatives we tried attaching a 3d-printed gripper but didn't work because of the slippery of the PLA material and the irregular form of the Robotiq gripper. We also tried attaching double-faced tape and Velcro stripes but in the moment of leaving the egg the tape wouldn't drop the egg in consequence it remain attached to the gripper. The most optimal solution was to make an gripper made from the fingertips of an Latex glove and coffee grains. This in order to make the grains adapt to the shape needed for the correct grip of the egg toy. The downside of this was it´s durability. The coffee grip had to be constantly changed in order to work efficiently, the change of said gripper had to be done approximately every 2 hours before the latex broke down.

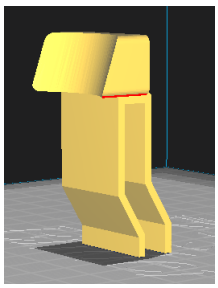


Fig. 30: 3D CAD of the trial of gripper for the egg

5.3.3 Assembly of the egg and delivery to the final robot

This part of the experiment was the most inconsistent, a lot of factors detonated the failure of many of the attempts, approximately the success rate was about 10 percent, the error from the puzzlebot position and the coordination of the arms was to daunting, even with experiments at different speed and different torque. The error accumulated in all the different parts of the experiments made this the most imprecise part of everything and also the most difficult. After that the robot had to put an egg basket in Motomami Puzzlebot, this was done swiftly with 100 percent success rate.

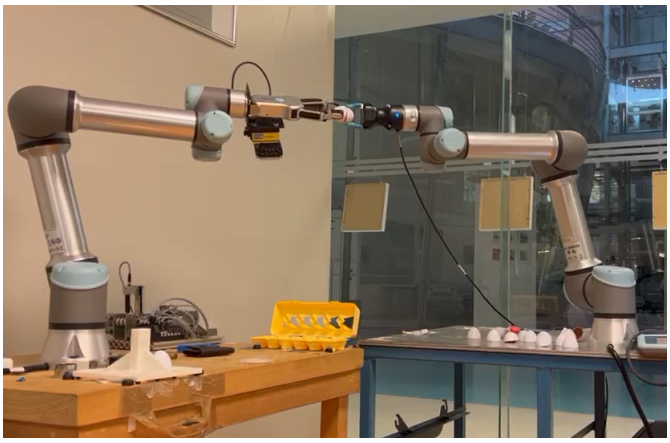


Fig. 31: Robots trying to assemble two eggs

6 Conclusion

Since both the delivering and the assembling sections worked independently, we can consider the implementation as a success, since both sections were inside the expected error range. But, given that there was a need to manually correct some aspects of the collaboration, there's certainly improvements to our implementation.

Given than our implementation was defined by the educational goals of the subjected attached to our project, some of the changes were not possible, such as the implementation of amcl. Another change would be the support of the wooden table of one of the robotic arms, since this generated changes in the position of the base of the arm, making difficult to perfectly align the two assembly positions of the UR5 robots.

But, even with these challenges, our implementation was able to generate a solid base for future projects that use the same hardware as we did or a similar environment, and, since we tried most of the feasible solutions for the hardware we used, we can define that we reached the limits of the devices, specially the ones of the mobile robots.

References

[1] BMW_group: Innovative human-robot cooperation in BMW Group Production. (2013). https://www.press.bmwgroup.com/global/article/detail/T0209722EN/innovative-human-robot-cooperation-in-bmw-group-production?language=en

[2] The_Robot_Remix: How Tesla Used Robotics to Survive Production Hell and Became the World's Most Advanced Car Manufacturer (2022). https://www.roboticstomorrow.com/article/2022/06/2022-top-article-how-tesla-used-robotics-to-survive-production-hell-and-became-\the-worlds-most-advanced-car-manufacturer/18908

[3] Tsubouchi, T.: Introduction to Simultaneous Localization and Mapping. Journal of Robotics and Mechatronics. (2019). https://www.researchgate.net/publication/333905097

[4] Zhang, S.S..N.P. T.: Review of recent advances in Simultaneous Localization and Mapping. Information Fusion, 36, 1-20. (2017)

[5] OpenSLAM.org.: OpenSLAM.org (2018). https://openslam-org.github.io/gmapping.html

[6] S. Souza, M.R..G.G.L.M. A. A.: 3D Probabilistic Occupancy Grid to Robotic Mapping with Stereo Vision. InTech. doi: 10.5772/49050 (2012)

[7] Jain, .A.R. R.: Navigation in unknown terrains: Path planning strategies for autonomous mobile robots. Journal of Robotic Systems, 12(7), 467-482. (2005)

[8] Cormen, L.C.E.R.R.L..S.C. T. H.: Introduction to algorithms. MIT press. (2009)

[9] Arkin, R.C.: Behavior-based robotics (2000)

[10] Robots, U.: UR5 Collaborative Robot (2023). https://www.universal-robots.com/products/ur5-robot/

[11] Robotics, R.: A Review of Universal Robots UR5. (2015). https://www.rethinkrobotics.com/blog/review-universal-robots-ur5/

[12] Mauch, F.: Universal Robots URCap ROS. (2015). https://github.com/UniversalRobots/Universal_Robots_ROS_as_a_Service_URCap

[13] NVIDIA: NVIDIA Jetson Nano System-on-Module Data-sheet (2022). https://developer.nvidia.com/embedded/downloads#?search=Jetson%20Nano

[14] DFROBOT: RPLIDAR A1M8-R6 - 360 Degree LiDAR Laser Range Scanner (12m) (2023). https://www.dfrobot.com/product-1125.html

[15] supply, P.: Camera Module for Official Raspberry Pi Camera Board V2 8MP Sensor - 160 Degree (2023). https://uk.pi-supply.com/products/camera-module-for-official-raspberry-pi-camera-board-v2-8mp-sensor-160-degree