Programación algorítmica de un brazo robótico

PROYECTO

presentado para optar

al Título de Grado en Ingeniería en Electrónica Industrial por

Analucía Vera Maric

bajo la supervisión de

"Iñaki Díaz"

Donostia-San Sebastián, mayo de 2023

Tecnun
Universidad de Navarra
ESCUELA DE INGENIERÍA
INGENIARITZA ESKOLA
SCHOOL OF ENGINEERING

La oportunidad que el CEIT me brindó me marcará el resto de mi vida profesional. Agradezco enormemente a la institución y particularmente a *Iñaki Díaz* y *Axier Ugartemendia* por encaminarme en este proyecto tan creativo y apasionante. Fueron esenciales en cada etapa, armándome de conocimiento, prestándome todas las facilidades posibles y colaborando a solucionar todos los problemas que surgieron.

Desearía a la vez, expresar mi profundo agradecimiento tanto a mis padres y como a mi hermana. Sin su ayuda y su mentoría no hubiera logrado cumplir todo lo que he cumplido hasta el día de hoy. Son en mi vida un modelo a seguir y espero estar a la altura de las expectativas que ven en mi futuro.

A nivel personal, este proyecto no sólo me brindó la oportunidad de descubrir más profundamente el mundo de la robótica, sino, me permitió también desarrollar mis habilidades como ingeniera.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1.INTRODUCCIÓN

1.1 Resumen

Este proyecto de fin de grado es una programación algorítmica de un brazo robótico, que se divide en dos programas. Dichos programas, serán ejecutados gracias a la ayuda de un brazo robótico (Omron TM5) que, en este caso, posee también una cámara de visión artificial 2D necesaria para el buen funcionamiento de ambos. Como primera misión, se busca simular la mente humana en lo que concierne la secuencias de toma de decisiones. Con la finalidad de lograr lo mencionado, se buscará conformar un sistema integrado del clásico juego de tres en raya, donde el robot se enfrentará a un usuario humano. Como segunda misión, el brazo robótico buscará reconocer rostros humanos para luego plasmarlos en una hoja de papel.

Tomando el primer programa bajo observación, previamente a la puesta en prueba del sistema de este, la cámara tendrá guardada en su memoria las diferentes formas de los objetos a ser utilizados para el juego, en este caso bloques (para el usuario) y cilindros (para el robot). El usuario deberá comenzar el juego posicionando el primer bloque en la plantilla. Una vez el bloque es reconocido y su posición matricial analizada, el brazo robótico junto con la cámara se desplazarán a la llamada "posición de agarre". Una vez instaurados en la "posición de agarre", el robot deberá identificar entre los cilindros colocados aleatoriamente el cilindro cuyo orden le corresponda, ya que estos se encuentran numerados del 1 al 4. Esto, con el fin de trasladar dicho cilindro y colocarlo en la plantilla en la posición que mejor nos convenga. Así, a través de esta memorización de los objetos y el análisis de las diferentes posiciones matriciales el robot desencadenará una serie de condiciones que lo llevará a la victoria o al empate. Recalcando lo previamente mencionado, la programación está realizada de tal manera que no es posible que el juego desencadene en la derrota del robot.

Sobre la segunda misión, el brazo robótico gracias a la cámara integrada identificará en un primer tiempo el contorno de la cara humana para después dibujarlo en una hoja de papel. Para dicho propósito, la cámara 2D estará constantemente midiendo las coordenadas y distancias entre diferentes puntos clave del rostro humano para establecer los límites de las líneas a ser marcadas. Estos puntos claves, estarán marcados con ocho formas diferentes colocadas en el rostro de un individuo que se posicionará frente al robot . Acto seguido, gracias a la memorización de una plantilla especial (figura 128), el robot será capaz de manejarse a través de la hoja de papel puesta ante él para dibujar el rostro.

1.2 Abstract

This final degree project is an algorithmic programming of a robotic arm, which is divided into two programs. These programs will be executed thanks to the help of a robotic arm (Omron TM5) which, in this case, also has a 2D artificial vision camera necessary for the correct operation of both programs. As a first mission, the aim is to simulate the human mind in terms of decision-making sequences. In order to achieve this, we will create an integrated system of the classic tic-tac-toe game, where the robot will face a human user. As a second mission, the robotic arm will seek to recognize human faces and then capture them on a sheet of paper.

Taking the first program under observation, prior to testing the system, the camera will have stored in its memory the different shapes of the objects to be used for the game, in this case blocks (for the user) and cylinders (for the robot). The user must start the game by positioning the first block in the template. Once the block is recognized and its matrix position analyzed, the robotic arm together with the camera will move to the so-called "gripping position". Once installed in the

"gripping position", the robot must identify among the randomly placed cylinders the cylinder whose order corresponds to it, as these are numbered from 1 to 4. This, in order to move the cylinder and place it in the template in the position that best suits us. Thus, through this memorization of the objects and the analysis of the different matrix positions, the robot will trigger a series of conditions that will lead it to victory or to a draw. It is worth specifying that the programming is done in such a way that it is not possible for the game to lead to the defeat of the robot.

Regarding the second mission, the robotic arm, thanks to the integrated camera, will first identify the outline of the human face and then draw it on a sheet of paper. For this purpose, the 2D camera will constantly measure the coordinates and distances between different key points of the human face to establish the limits of the lines to be marked. These key points will be marked with eight different shapes placed on the face of an individual who will be positioned in front of the robot. Then, thanks to the memorization of a special template, the robot will be able to move across the sheet of paper placed in front of it to draw the face.

1.3 Justificación

El planteamiento del presente proyecto de fin de grado tiene como propósito adentrarnos en la programación de los sistemas robóticos desafiando los límites para los cuales estos fueron creados, estos últimos siendo tareas repetitivas sin necesidad de mayor análisis. Más específicamente, los brazos robóticos son creados con la sola función de hacer movimientos repetitivos con un código ligero, siendo esto ajeno a lo propuesto. Basándonos en las diferentes posibilidades que estos sistemas nos brindan, buscamos adquirir todas las competencias posibles para establecer ideas de cambios futuros que mejoren e innoven en el ámbito de la robótica.

La puesta en práctica de proyectos que estén ligados a la ejecución de sistemas robóticos, brinda la oportunidad de encararse de manera directa con los problemas y dificultades característicos de esta área. Así mismo, abre la puerta a la creatividad humana para resolverlos de manera práctica ante cualquier imprevisto.

1.4 Objetivos

Los fines de creación de los robot implementados en la industria, residen en la repetición de tareas y movimientos que no necesiten mucha meditación. El objetivo del proyecto reside en desarrollar la mente autodidacta para aprender nuevas interfaces con el propósito de realizar distintas aplicaciones que excedan los fines de creación de dichas interfaces. Demostrar al usuario el alcance que puede llegar a tener una de las tecnologías más simples de automatización de procesos.

Bajo la finalidad de llevar a cabo las dos misiones previamente mencionadas, se ejecutaron una serie de metas:

- Estudio del funcionamiento del brazo robótico y de sus periféricos.
- Comprender la interfaz del sistema y familiarizarse con él.
- Creación de códigos básicos con fin de entender los posibles errores e imprevistos.
- Planificación del camino que el código ha de seguir.
- Programación del robot.
- Análisis del funcionamiento y de los errores presentes en el desempeño de la programación del robot.
- Corrección del mal funcionamiento del programa.

1.5 Aplicaciones

Notablemente, las aplicaciones para los sistemas derivados de la coordinación entre cámaras con visión artificial y robots son inmensurables. Dichos sistemas pueden verse aplicados claramente tanto en el sector alimenticio como el sector industrial, donde son utilizados para reemplazar tareas o procesos que pueden poner en peligro la salud del ser humano.

Llevar dicha coordinación a niveles superiores de su predisposición supone una innovación en términos de precisión y manipulación de los sistemas. Una diversificación de las programaciones posibles puede ser de gran aporte a largo plazo ya que significaría un cambio paulatino de las posibilidades que pueden llegar a tener los robots con dicha característica. Más específicamente, dichos programas brindarán nuevas oportunidades para las creaciones de bienes que excedan la capacidad humana.

2.OMRON TM5

2.1 Introducción a la robótica

2.1.1 Conceptualización

Antes de toda inmersión sobre la robótica, es preciso tener en mente las definiciones y conceptos de qué es la robótica industrial y cuáles son sus definiciones. Presentando un acercamiento más general, se puede conceptualizar la robótica industrial como una rama de las ingenierías mecánica, electrónica y ciencias computacionales, dedicada al estudio, desarrollo y programación de robots.

La etimología de la palabra "robot" se presta del inglés, que, a su vez, es extrapolado del checo denominado como "robota" significando trabajo pesado. Con el propósito de profundizar el asunto, asumimos la definición de la palabra "robot" establecida por la RIA, Robot Industry Association, "un robot industrial es un manipulador reprogramable multifuncional diseñado para mover materiales, piezas, herramientas o artefactos especiales, mediante movimientos variables programados, para la ejecución de tareas potencialmente muy diversas."

2.1.2 Evolución histórica

La robótica industrial es una tecnología que se ha desarrollado a lo largo de los últimos ochenta años, logrando en su trayectoria, transformar tanto el mundo de la industria como la vida laboral de los individuos en variados ámbitos.

Como primera aproximación a lo que hoy conocemos como robots industriales, se creó en 1937 la Gargantúa. Dicho robot se creó bajo el propósito de colocar productos en espacios especiales, lo que hoy se conoce como pick and place. La Gargantúa era capaz de apilar bloques de madera bajo patrones programados, dando así, inicio al mundo de la programación robótica. En los años siguientes a la Gargantúa, se empezaron a desarrollar y comercializar robots con fines industriales y de ocio.

A mediados del siglo XX, tras la culminación de la Segunda Guerra Mundial, los primeros robots industriales de gran tamaño fueron puestos a prueba. Es así, como a principios de los años sesenta la compañía estadounidense Unimation dirigió el desarrollo del primer robot de transferencia programable, considerado el embrión del robot industrial conocido en la actualidad. La inspiración infundada por Unimation, dio paso a una de las empresas pioneras del uso de la robótica (General Motors) para que lleve a cabo la producción del primer brazo robótico implementado en las primeras cadenas de montaje automatizadas.

Años más tarde, cerca de la década de los setenta, Unimotion se extendió a Europa, momento en el cuál Nokia lideraba la implementación de robótica industrial. En este periodo de tiempo, Nokia sacó al mercado los robots de soldadura, revolucionando así, el mundo del montaje ya que tenían la capacidad de soldar y montar 110 automóviles por hora.

Posteriormente, en 1975 se fundó en Europa el ASEA IRB. Éste fue el primer robot totalmente eléctrico acondicionado bajo la tecnología intel. Dos años después, Unimotion y Vicarm, fabricaron y comercializaron el robot PUMA (Programable Universal Machine for Assembly). El robot PUMA diseñado por el inventor del Standford Arm, Victor Scheinman, se utilizaba como uno de los actores en las líneas de ensamblaje de General Motors realizando tareas de pick and place. Adicionalmente,

este brazo robótico fue utilizado como sujeto de investigación dadas las posibilidades que ofrecía su lenguaje de programación.

Los años ochenta fueron críticos para el desarrollo de la robótica. La década de los ochenta es considerada como el inicio de la era de la robótica dado que su venta y producción se vieron aumentados por casi el 80%. Se tiende a estimar que en esta época se empezaron a establecer los cimientos que harían surgir la robótica inteligente. Gracias a la evolución de la tecnología y la implementación de la inteligencia artificial los robots adquirieron independencia.

Los eventos previamente mencionados, forman parte de la historia de las pasadas revoluciones industriales que nos ayudan a definir tanto la situación actual como las características que envuelven el mundo de la robótica. Adentrándonos superficialmente a las aportaciones de las revoluciones industriales, podemos distinguir que cada una se caracteriza por la introducción o uso de alguna técnica o material innovador que permite avances en el sector. Es gracias a estos avances que la presente revolución industrial, denominada era de la industria 4.0, se puede caracterizar notablemente por la utilización de robots industriales.

2.1.3 Descripción del brazo robótico

El brazo robótico, como principal misión, busca ayudar al operario en tareas tanto repetitivas como peligrosas, como es el caso de varios centros de montaje con maquinaria pesada. Bajo la lupa inspectora del mundo de la robótica industrial, el brazo robótico debe cumplir ciertas características que refuercen su función de brindar seguridad al operario al momento de tratar piezas. Dichas características son:

- El cumplimiento de las normativas al límite de exposición de los operarios a campos electromagnéticos.
- Diseño compacto para favorecer el transporte.
- Diseño económico, componentes de fácil adquisición en el mercado en caso de avería.
- Alta automatización para facilitar su implementación y reducir tanto errores como costes de funcionamiento.

El robot puesto en práctica en el presente proyecto cumple todas las características previamente mencionadas teniendo como principales componentes la base móvil del robot, el brazo articulado (figura 1), la cámara 2D (figura 2) y la garra (figura 3).

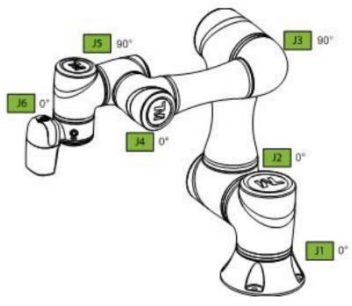

Figura 1: Brazo articulado

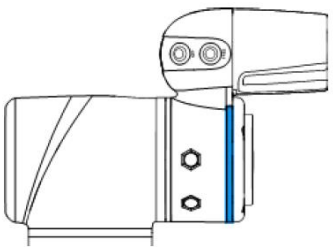

Figura 2: Cámara 2D

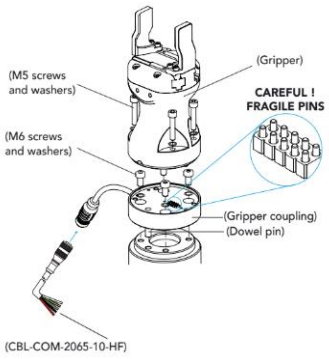

Figura 3: Garra

De manera general, los elementos que conforman nuestro robot son los siguientes:

- Actuadores: mecanismos que conceden un movimiento articulado al robot (ilustrados en la figura 1). Estos mecanismos son sistemas que funcionan ya sea de manera hidráulica, eléctrica o neumática. En nuestro caso, podemos hacer alusión al sistema de funcionamiento eléctrico ya que son utilizados servomotores.
- Brazo mecánico (figura 1): elemento conformado por componentes rígidos que, unidos forman articulaciones. Gracias a este elemento, se logrará posicionar la herramienta a ser utilizada en la posición adecuada.
- Sensores: los sensores buscan comunicar al sistema de control y al operario el estado del robot. Dentro de los sensores, podemos distinguir dos tipos de sensores los externos e internos. Un tipo de sensor presente en nuestro robot serán los sensores internos, *encoders*. Sensores que miden constantemente la posición angular de las articulaciones. Lamentablemente, en nuestro robot no se distinguen sensores externos.
- Microcontroladores: supervisan y controlan el movimiento del robot.

2.2 Configuración y programación

Nuestro robot tiene por sistema de programación uno desarrollado por la misma empresa creadora, Omron. TMFlow es un sistema HMI (Human-Machine Interface) que se basa en la programación gráfica. Como lo establece en su manual, el entorno de programación de los robots Omron busca brindar al usuario simplicidad a la hora de programar y definir parámetros, tomando como base los diagramas de flujo.

Bajo el propósito de llevar a cabo una correcta programación y aplicación del robot, debemos tener presentes los conceptos básicos del robot, así como, estar familiarizados con las diferentes interfaces del robot.

En primer lugar, nos enfocaremos en las interfaces externas del robot. Más específicamente, en esta primera aproximación a la configuración del robot buscaremos entender los diferentes paneles de control y botones externos del mismo. El robot tiene conectado a sí mismo un control con funciones básica, conteniendo a su vez un botón de emergencia en caso de violación de seguridad.

Figura 4: Control del robot

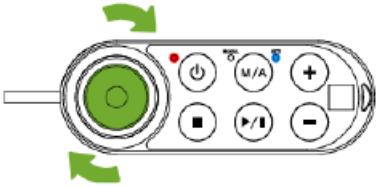

Figura 5: Botón de emergencia

Adentrándonos en las funciones presentes en el control, podemos explicar la funcionalidad de cada uno de los botones de la siguiente forma:

- Botón de encendido (⏻): al ser presionado puede encender o apagar el robot.
- Botón manual/automático (M/A): al iniciar el robot, este empieza estando en la configuración automática (LED azul). Dicha configuración no nos permite editar o mover libremente el robot, por lo que tendremos que cambiarlo a la configuración manual (LED

verde). No obstante, para cambiar de manual a automático tendremos que presionar el botón M/A seguido de la siguiente secuencia: + - - + -.

- Botones +/-: estos botones ayudan a aumentar o disminuir la velocidad de aplicación del robot respectivamente. Adicionalmente, como mencionado anteriormente, estos botones ayudan también a cambiar de configuración manual a automática.
- Botón de parada (⏹): al presionar este botón el robot para la simulación que esté llevando a cabo.
- Botón reproducir/pausar: para o reanuda la simulación que el robot esté poniendo a prueba.
- Botón de emergencia: ha de ser presionado una vez el robot lleve a cabo un movimiento que suponga un riesgo a otros operarios o a sí mismo.

Además del control, el robot tiene presente otros botones posicionados al costado de la cámara integrada. El botón de la izquierda nos permite guardar la posición actual del robot como un punto en el programa, mientras que, el botón de la derecha (figura 6) nos permite mover libremente el robot mientras este siga estando presionado.

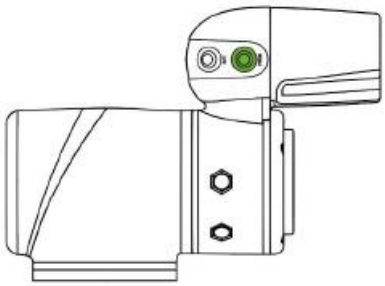

Figura 6: Botón de libre movimiento

Como se ha mencionado previamente, si el robot se encuentra en configuración automática no será posible programarlo ni manipularlo libremente. Para poder distinguir claramente la configuración del robot, se disponen diferentes LEDs de advertencia (figuras 7 y 8).

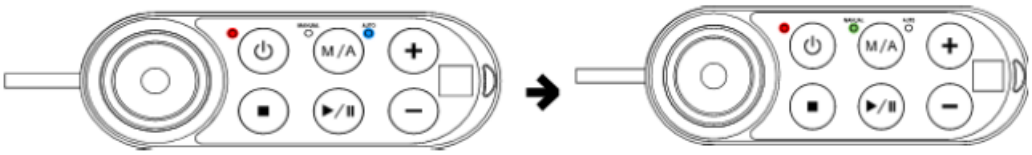

Figura 7: Modo automático/Modo manual LEDS control

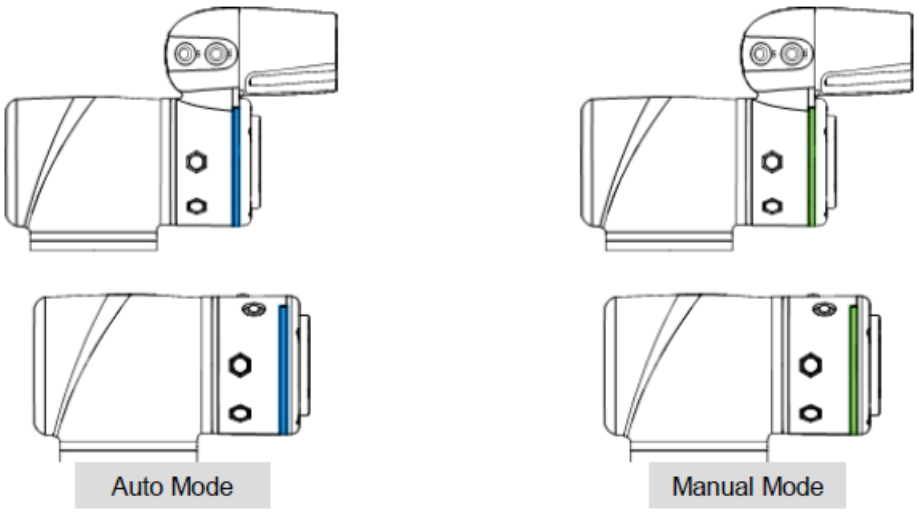

Figura 8: Modo automático/Modo manual robot

En segundo lugar, nos enfocamos en aspectos internos de la interfaz que deberán tenerse en cuenta a lo largo de la programación. Para poder conectarnos al robot de manera correcta tendremos que iniciar sesión en la interfaz TMFlow. Una vez iniciada la sesión (figura 9), pulsaremos el botón en pantalla que establece "get permission" (figura 10) para lograr establecer el lazo entre el robot y la programación.

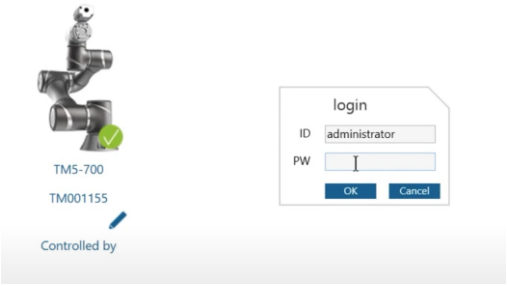

Figura 9: Login

Figura 10: Establecer lazo

Al presionar ≡ tendremos acceso a un menú que se desplegará a la izquierda de la pantalla. En dicho menú encontraremos las siguientes opciones:

- Login/Logout: necesario solo al momento de establecer el lazo entre el robot y la programación.
- Connect: brindará una lista de dispositivos conectados.
- View: un menú con diferentes vistas y controles del robot una vez este esté funcionando.
- Run setting: lista de proyectos guardados en la memoria.
- Project: lugar en el que podemos modificar el proyecto que está siendo ejecutado.

- Setting: la configuración del robot.
- System: la configuración del sistema del robot.
- Shutdown: Apagado del robot a través del programa.

Los pasos descritos previamente serán los que se tendrán que llevar a cabo cada que se reinicie el funcionamiento del robot. Adicionalmente a lo explicado, podemos profundizar las aplicaciones pertenecientes a "View" dado que serán de gran ayuda a la hora de la programación general del sistema.

Una vez nos encontramos en "View", podemos observar siete opciones diferentes de supervisión de la ejecución del programa. Dichas opciones pueden ser descritas de la siguiente manera:

- Display board: lugar en el cual podremos observar tanto lo que la cámara captura una vez esta sea utilizada como el texto que sea programado bajo las funciones de "display".
- Flow: seguimiento del programa mientras este está en ejecución.
- IO: monitorea el estado de los inputs/outputs del sistema.
- Simulator: monitorea y muestra en pantalla un modelo virtual del robot en su posición actual con los valores de las rotaciones de las articulaciones.
- Status: valores que toman distintas variables necesarias para el estado del robot. Entre las variables se pueden distinguir la temperatura, el voltaje, la potencia, la corriente del robot, etc.

Dado que a lo largo de la programación estaremos trabajando con diferentes bases, estas principalmente introducidas por el elemento de visión, enfatizamos la presencia del elemento en pantalla que nos expresa la base en la que estamos trabajando actualmente (figura 11 y 12).

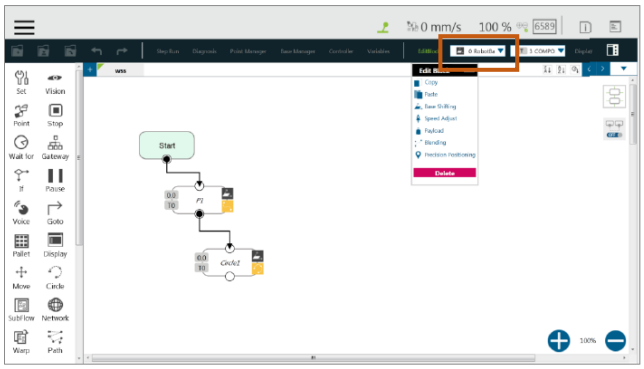

Figura 11: Pantalla de trabajo

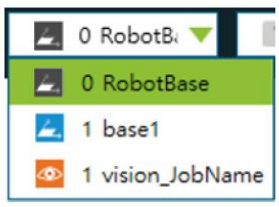

Figura 12: Base actual del robot

Otros elementos de la barra de configuración que debemos tomar en cuenta son "Point Manager" y "Variables". Dentro de "Point Manager" podremos visualizar todos los puntos guardados en memoria. El que un punto o alguna variable esté marcado con un signo de exclamación indica que este punto o variable no está siendo utilizada en el programa (figura 13). Una vez nos encontremos en "Variables" tendremos la capacidad de crear y/o borrar variables (figura 14). Es importante recordar que las variables introducidas en el código no podrán ser monitoreadas tanto en su ejecución como una vez terminado el programa. Para poder analizar los valores de las variables es necesario crear variables globales que guarden los valores de las variables locales. La opción de variables globales se encuentra dentro de la configuración del robot (figura 15).

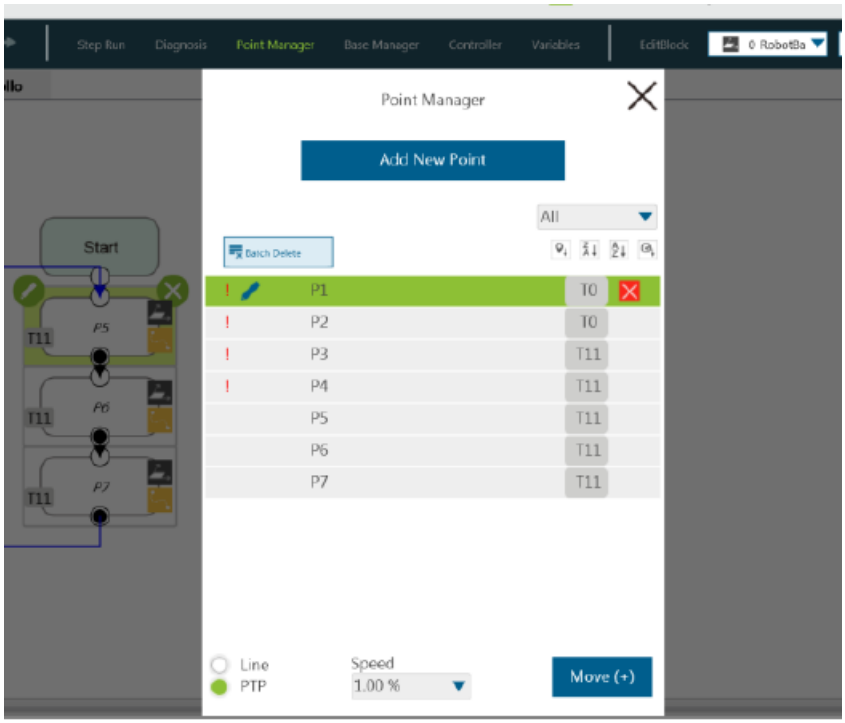

Figura 13: Point Manager

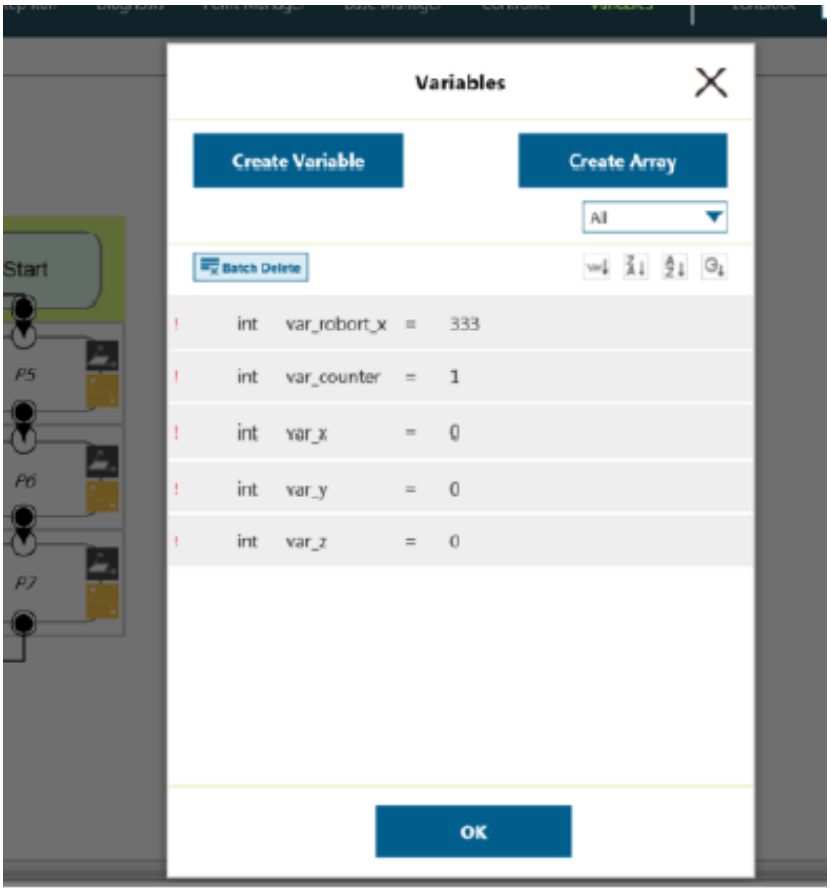

Figura 14: Variables

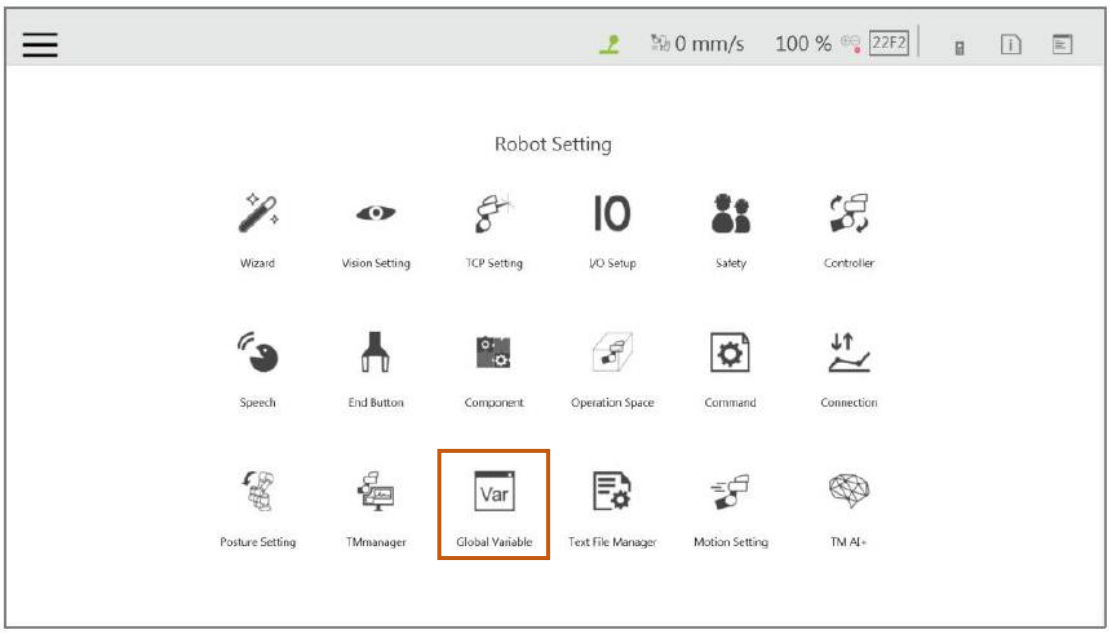

Figura 15: Variables globales

Considerando que nuestra programación estará implementada por nodos básicos, no sobra explicar brevemente la funcionalidad de estos y cómo se utilizan. Los nodos que más se utilizarán en la programación serán los de "Point", "Move", "Set" e "If".

El nodo "Point" nos permitirá registrar un punto en el espacio, guardando las posiciones de las articulaciones. Este nodo, puesto que es un nodo de movimiento, contiene tres formar diferentes de desplazarse, "Point To Point", "Line" y "WayPoint". Como explicación gráfica del desplazamiento de cada una de ellas se puede tomar como referencia las figuras 16, 17 y 18 respectivamente. Dado que trabajaremos con objetos tridimensionales, a fin de evitar colisiones, cada que usemos la función "Point" procuraremos definir el desplazamiento como "Line" o "WayPoint".

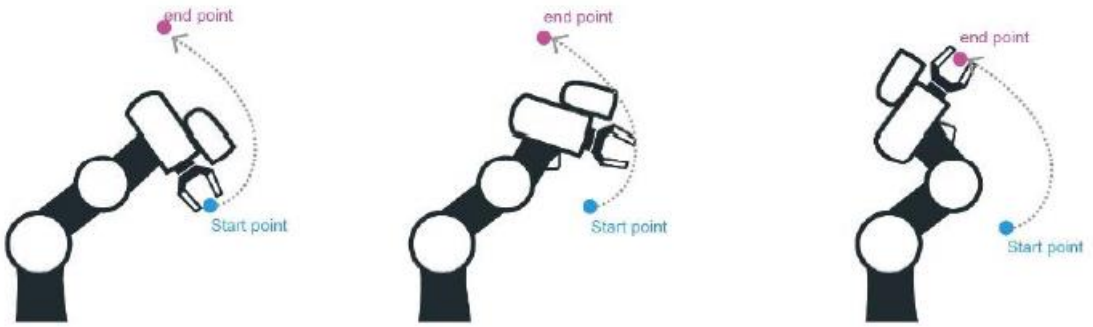

Figura 16: Movimiento Point To Point (PTP)

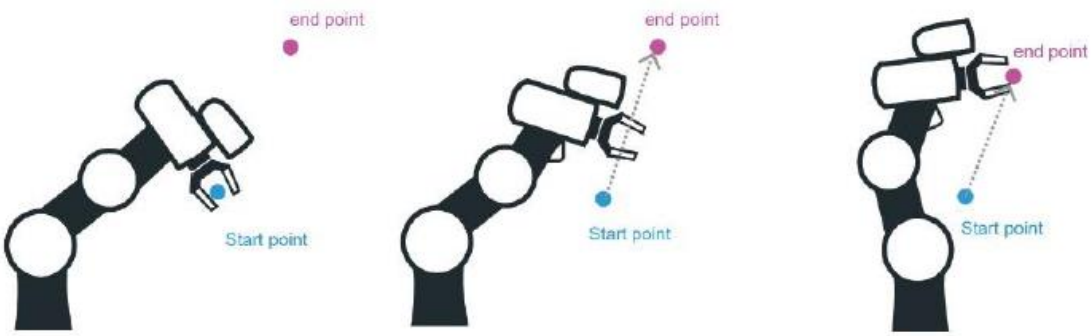

Figura 17: Movimiento Line

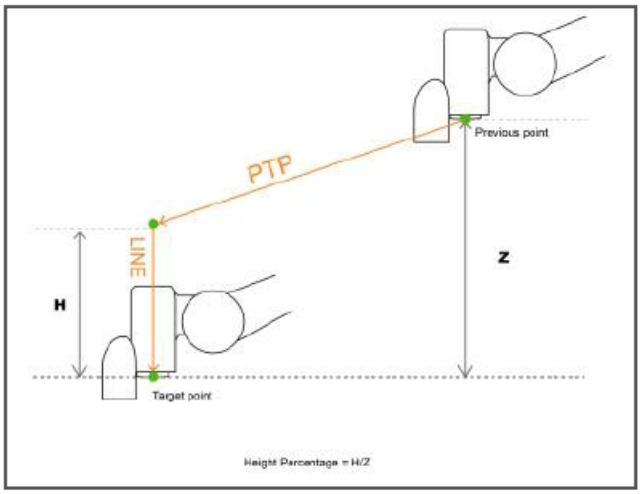

Figura 18: Movimiento WayPoint

Paralelamente al nodo "Point", el nodo "Move" nos permitirá mover el robot ya sea por valores en los ejes o por rotaciones de las articulaciones. Para mayor simplicidad de diseño, el movimiento del robot se basará en la aplicación de coordenadas. Puesto que este nodo nos permite seleccionar las variables que tomará como referencia de valor para su movimiento, centraremos la programación en la definición de variables para facilitar el movimiento del robot. Como mencionado previamente, al trabajar con objetos tridimensionales, programaremos los movimientos con un tipo de desplazamiento linear marcando la opción "Line".

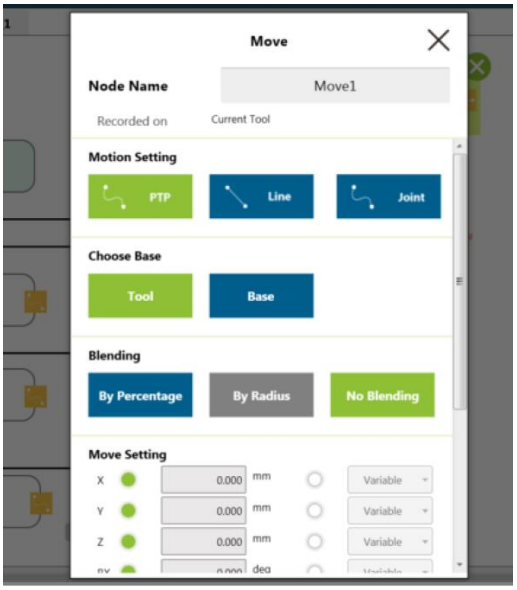

Figura 19: Nodo de movimiento

Los nodos que quedan por presentar son "Set" e "If". El nodo "Set", por su parte, nos permite establecer los valores de las variables que vayamos a utilizar. Por otra parte, el nodo "If" nos da la oportunidad de introducir una condición dentro del programa, dándonos a la vez la opción de tener todas las condiciones en consideración o sólo una de ellas (Figura 20).

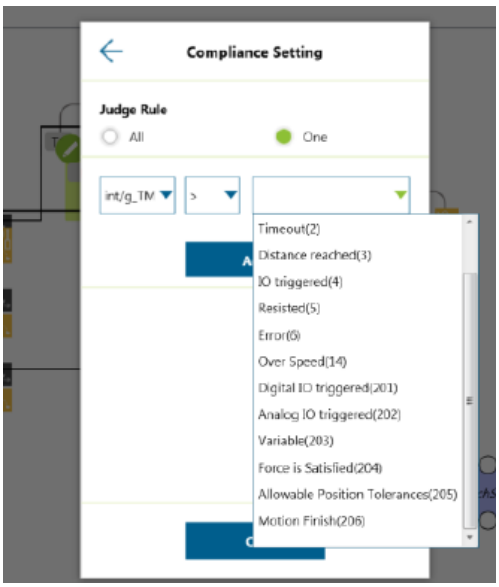

Figura 20: Configuración del nodo de condición

Finalmente, una función conocida generalmente en el mundo de la programación como debugger en inglés o depurador en español, puede ser encontrada en la barra principal del espacio de programación bajo el nombre de "*Step Run*". Esta función nos ayudará a ejecutar el programa paso a paso, o como en este caso, bloque a bloque.

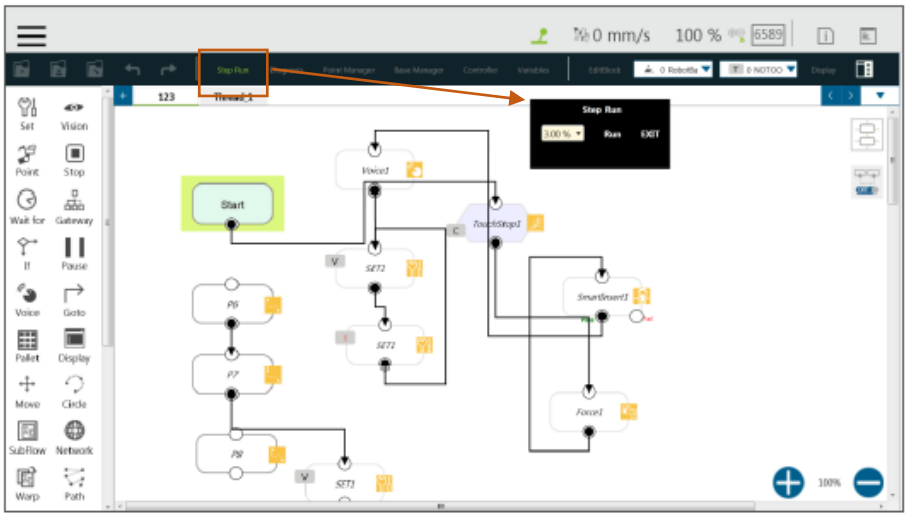

Figura 21: Step Run

3.TRES EN RAYA

3.1 Inteligencia artificial

La inteligencia artificial, según Oracle Cloud, se refiere a sistemas o máquinas de imitan la inteligencia humana para realizar tareas siendo que estas pueden mejorar iterativamente a parte de la información que recopilan. No obstante, según el punto de vista de TechTarget, la inteligencia artificial es la simulación de procesos de inteligencia humana por máquinas, siendo estas especialmente sistemas informáticos. Como tercer punto de vista, podemos recalcar el de SAS (Analytics Software & Solutions), que explica que la inteligencia artificial hace posible que las máquinas aprendan de la experiencia, se ajusten a nuevas aportaciones y realicen tareas como seres humanos. Siguiendo este punto de vista, podemos presentar como notables ejemplos computadoras que son capaces de jugar ajedrez o incluso automóviles de conducción autónoma.

Puesto que una definición completa de lo que es la inteligencia artificial no está presente en actualidad, se pueden definir cuatro enfoques de esta disciplina. Los primeros dos enfoques se centran tanto en los sistemas que piensan como humanos, como en sistemas que actúan como humanos. No obstante, los últimos dos buscan focalizarse entrono a la racionalidad siendo estos sistemas que piensan racionalmente o actúan racionalmente.

El programa que será descrito y desarrollado bajo el nombre de tres en raya puede pintarse bajo la definición de inteligencia artificial. Dicho programa buscará actuar como un ser humano a la hora de tomar decisiones paso a paso una vez sea su turno, esto con la finalidad de nunca perder.

3.2 Programa

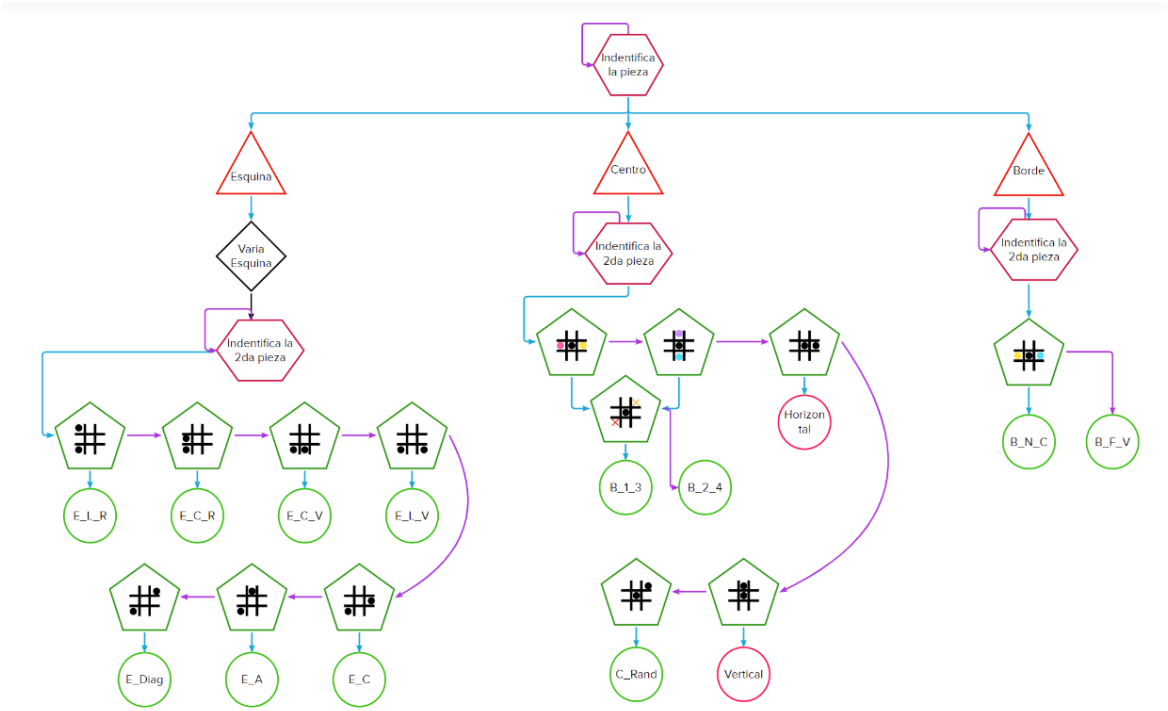

Figura 22: Diagrama de flujo del primer programa

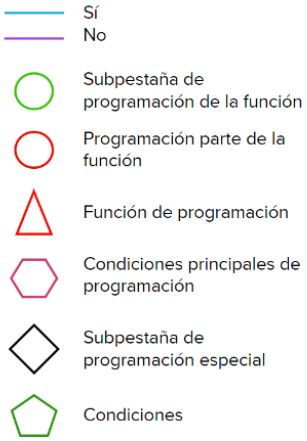

Figura 23: Leyenda del diagrama de flujo del primer programa

El programa de tres en raya se realizará de la manera descrita en el diagrama de flujo (figura 22). Explicando más detalladamente los pasos a seguir del sistema, encontramos cinco tipos de "funciones" o subpestañas de programación. Principalmente podemos encontrar lo que en programación se denomina como el "main" o pestaña principal de programación. Dentro de dicha pestaña podremos encontrar tres principales túneles redirigidos a funciones de programación. Estas pestañas como ilustradas anteriormente tienen por nombres: Centro, Esquina y Borde, representando así la situación inicial en la que el robot se encuentra una vez el usuario haya posicionado su primera pieza y esta haya sido reconocida.

Una vez nos encontramos en las funciones previamente nombradas, se desencadenará una serie de condiciones que establecerán el caso en el que nos encontramos una vez la posición de la segunda pieza del usuario sea identificada. Gracias al sistema mencionado, el programa nos llevará a subpestañas de programación en las que se desarrollará el final de la programación, teniendo en cuenta todas las combinaciones posibles.

Si bien el programa está descrito a grandes rasgos en el propuesto diagrama de flujo, es necesario detallar y explicar la funcionalidad y el proceso de programación de las partes que lo conforman.

3.2.1 Calibración y programación de la cámara

Puesto que en el programa que desarrollaremos la identificación de piezas es el elemento principal y uno de los más importantes, explicaremos cómo se llevó a cabo las diferentes funciones de la cámara.

Dado que la cámara con la que trabajamos contiene ciertas limitaciones, establecimos un orden tanto para los bloques a ser puestos por el usuario como para los cilindros a ser trasladados por el robot. Considerando que el primer paso a ser realizado por la cámara consiste sólo en identificar los objetos, no es necesaria una calibración previa al funcionamiento de esta. No obstante, reflexionando sobre el hecho que la cámara presentará un cierto error, comenzaremos realizando la calibración de la cámara posicionándola a una distancia del espacio de trabajo que nos brinde un error mínimo, pudiendo a la vez, observar a través de la cámara todo el espacio de trabajo.

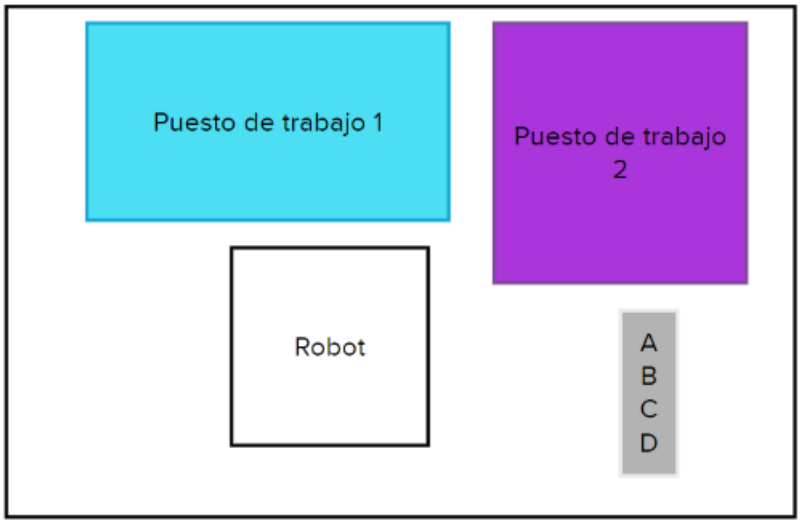

Figura 24: Disposición del espacio de trabajo

Nota: el robot tendrá dos puestos de trabajo. El primero será dónde se lleve a cabo el juego de tres en raya, mientras que, el segundo será el puesto del que el robot recoja los cilindros. El rectángulo gris, por su parte, representa la posición en la cual los bloques se encuentran inicialmente.

La programación de la cámara se accede a través del bloque perteneciente al menú contenido en "*Project*" llamado *visión* (figura 25). Una vez arrastrado a la sección de trabajo, notaremos que tiene un ícono a su derecha indicando la base de referencia que se está tomando (figura 26). Una vez entramos a su menú de configuración, encontraremos variadas opciones de reconocimiento y uso de la cámara. Esto último se puede ver ilustrado en la figura 16, en la cual estará resaltada la opción que utilizaremos.

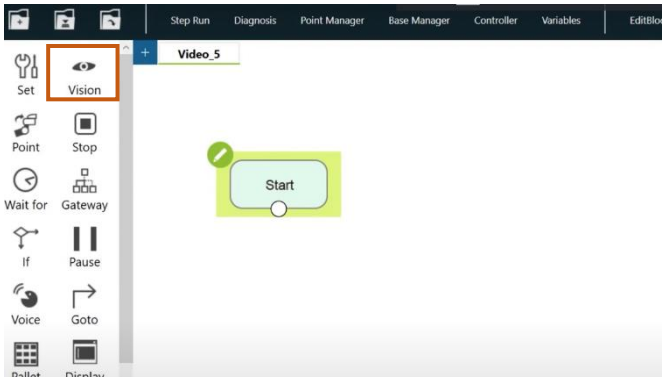

Figura 25: Menú Project, vision

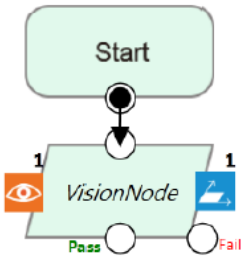

Figura 26: Nodo vision

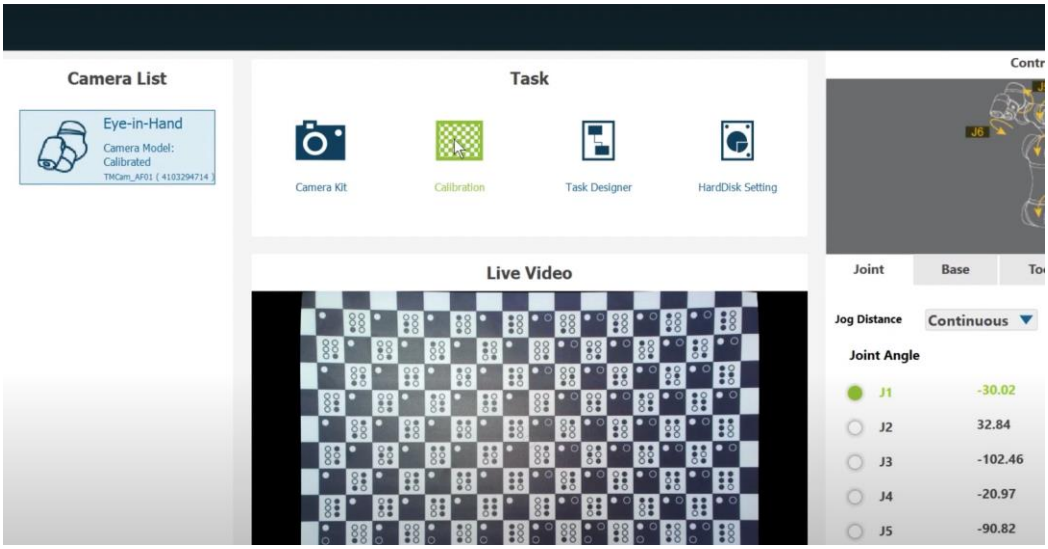

Figura 27: Menú cámara, calibración

Analizando al figura 27, podemos notar que en el cuadrado notado "Live Video" se puede ver la imagen capturada por la cámara a tiempo real. Dentro del recuadro notamos lo que llamaremos la placa de calibración. La placa de calibración es una plantilla proporcionada por Omron para realizar una calibración sencilla y fácil de la cámara del robot. Luego de seleccionar la opción resaltada, nos encontraremos con un menú con dos opciones, la calibración automática o manual. Una vez seleccionada la calibración automática, nos encontraremos con una serie de órdenes a ser realizadas para una correcta calibración de la cámara.

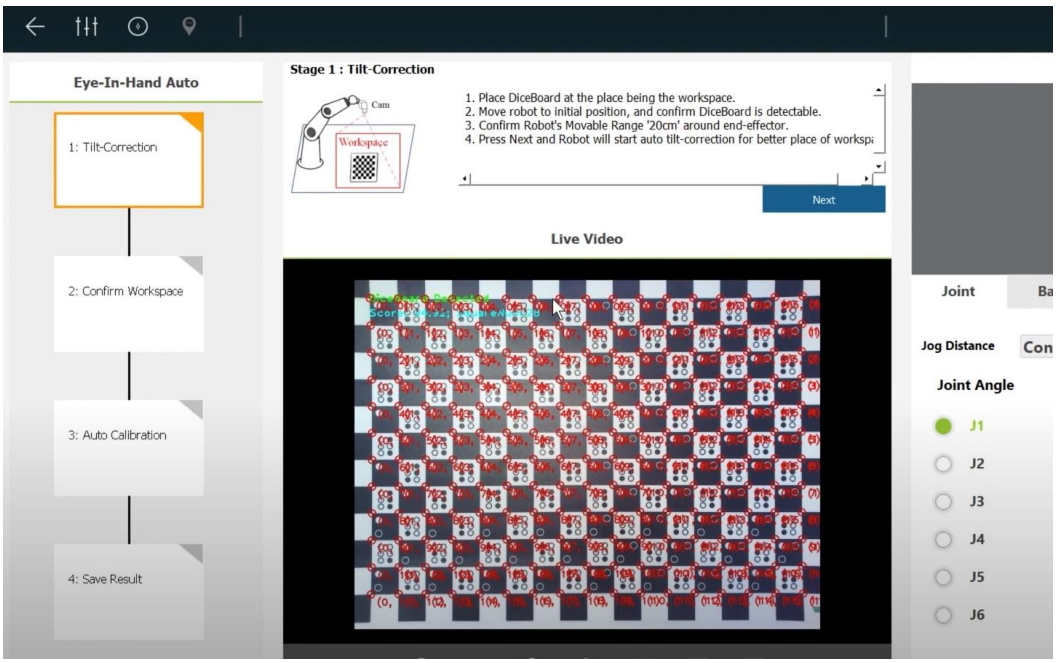

Figura 28: Calibración de la cámara

Después de la finalización de la calibración de la cámara, saltará en pantalla un aviso que indicará el error presente en la cámara. Para poder utilizar la cámara de manera apropiada, es imperativo realizar una calibración con un error menor al 0.5.

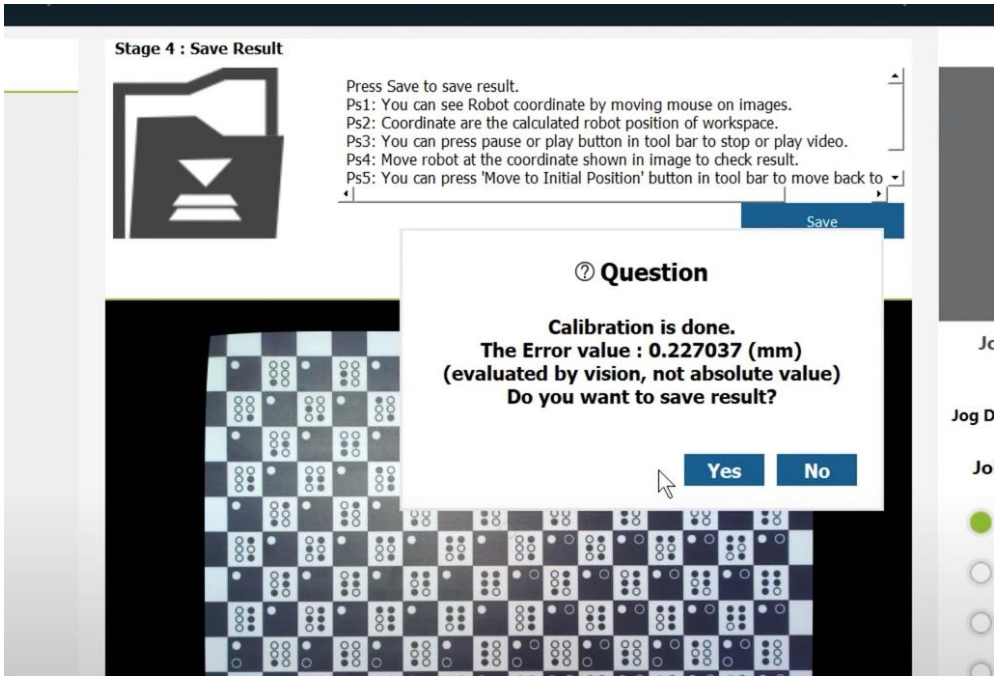

Figura 29: Error de la cámara

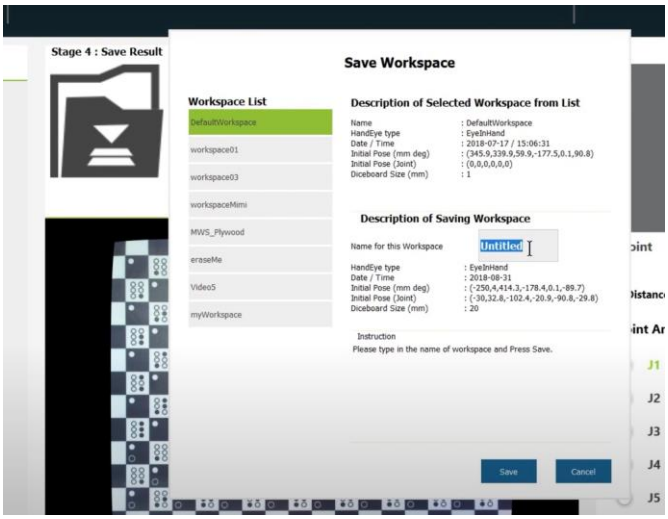

Figura 30: Guardado de la calibración

Nota: en la figura 29 notamos un error de aproximadamente 0.2207, no obstante, el error mencionado no es el error con el que trabajaremos. Nuestro error de calibración será de aproximadamente 0.31 tanto para el primer puesto de trabajo como para el segundo.

Una vez concluida la calibración de la cámara, volveremos al menú mostrado en la figura 27. A continuación de la calibración de la cámara bajo la referencia del primer puesto de trabajo, seleccionaremos la opción “Task Designer”. Esta opción nos llevará a otro menú en el cual

reconoceremos algunas de las opciones que utilizaremos a futuro. Por lo pronto, nos centraremos en la opción llamada "AIO-only" como lo muestra la figura 31. Dentro del apartado "AIO-only", escogeremos el ícono  para acceder a las opciones que nos permiten reconocer objetos. Dentro de las opciones que serán mostradas en pantalla, podemos destacar las siguientes:

- Pattern Matching (Shape): nos ayudará a localizar objetos basándose en su figura geométrica dándonos como salidas las coordenadas X, Y y el ángulo de rotación R.
- Pattern Matching (image): misma función que el anterior, solo que, basándose en la imagen en lugar de la figura geométrica.

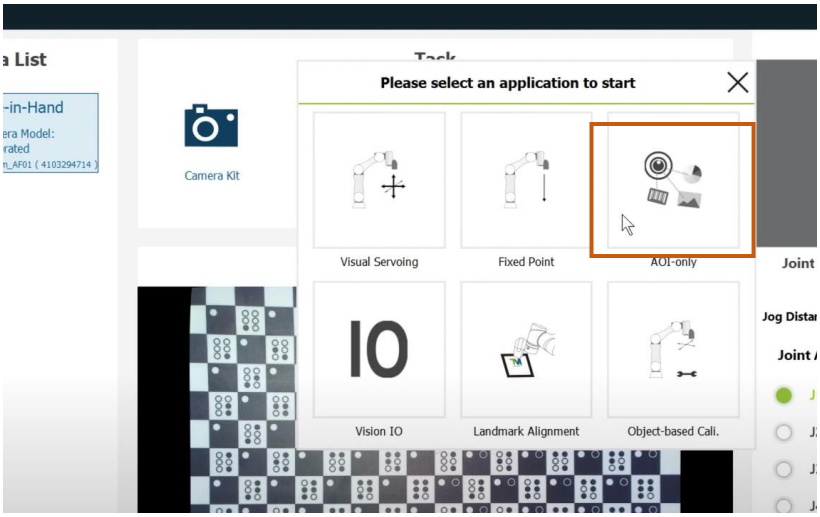

Figura 31: AOI-only

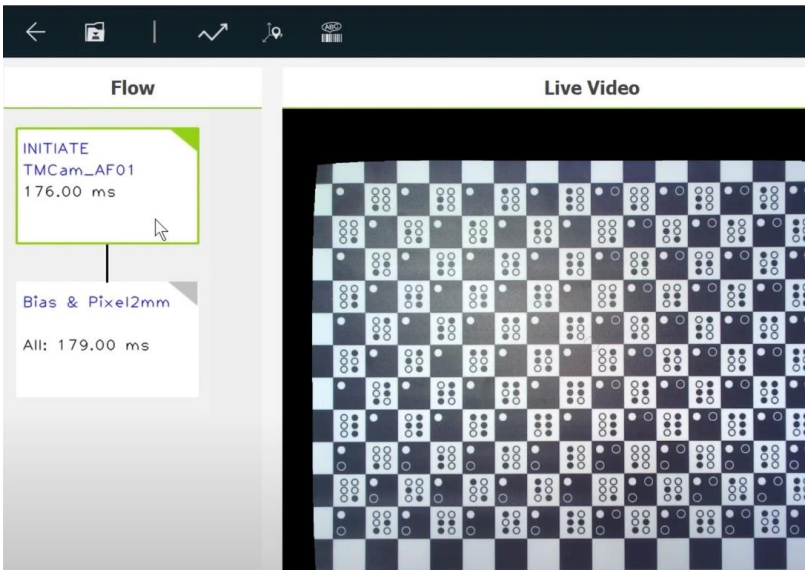

Figura 32: Función dentro de AOI-only

Bajo el propósito de generar la identificación más exacta posible, seleccionaremos "Pattern Matching (Shape)". Esta opción, como explicado previamente, nos permitirá basarnos en la figura geométrica del objeto con el que estemos trabajando, por lo cual, el reconocimiento de dicha pieza estará sujeto al contraste entre la figura y el fondo. Como se muestra en la figura 33, los bloques

están marcados con letras, mientras que, los cilindros están marcados con números. Será el contraste entre el fondo metálico de la pieza y la cinta negra el que delimite la figura geométrica del objeto a ser reconocido.

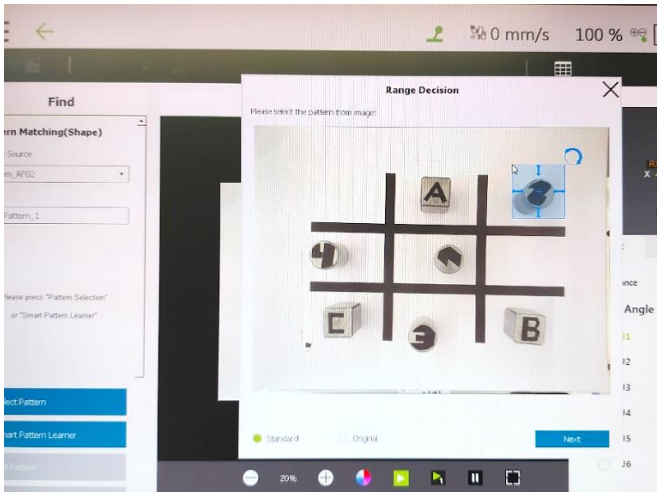

Figura 33: Figuras en cámara

Una vez guardada la geometría del objeto en cuestión, es importante notar que los valores devueltos por "Pattern Matching (Shape)" estarán en pixeles en lugar de milímetros. Para poder cambiar esta configuración tendremos que seleccionar "FINISH" (figura 34). Una vez estemos dentro de la configuración cambiamos los valores de retorno de pixeles a milímetros. A fin de conseguir una conversión exitosa, se tendrá que usar la placa de calibración y seguir los pasos mostrados en pantalla. Nótese que este paso de conversión tendrá que realizarse de manera individual por cada objeto que identifiquemos.

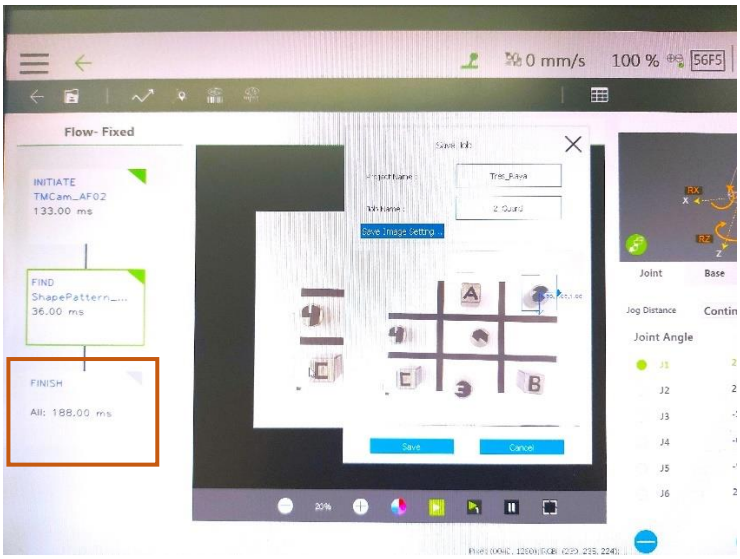

Figura 34: FINISH

Después de terminar de guardar en la memoria las diferentes formas de los objetos a ser manipulados, cuatro cilindros y cuatro bloques, conectamos el nodo visión al resto del programa. Es importante notar que el nodo de visión (figura 26) tiene dos puntos de conexión: pass y fail. Al

punto de conexión de la izquierda (pass) conectaremos el resto del programa, mientras que, al punto de conexión de la derecha le conectaremos el nodo "Goto", conectando este al mismo nodo de visión (figura 35). Esto último, bajo el principio que, si por alguna razón fallase el nodo de visión, por ejemplo, que la pieza no sea puesta, este siga intentando identificar el objeto de forma continua.

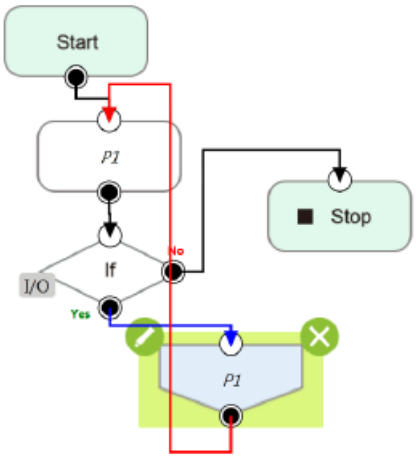

Figura 35: Ejemplo conexión nodo Goto

Dada la finalización de la calibración e identificación de los objetos en el primer puesto de trabajo, realizamos la calibración y la programación de las funciones necesarias para el segundo puesto de trabajo (figura 24).

El proceso de calibración de la cámara para el segundo puesto de trabajo se realizará de la misma manera que para el primer puesto de trabajo, ya que, lo único que difiere entre ambas es la orientación. Cabe a recalcar la importancia de la base de referencia actual del robot puesto que, en caso de realizar las calibraciones o identificaciones en base a otra que no sea la base de defecto del robot, podrían presentarse errores en su ejecución. Una vez guardada correctamente, nos volveremos a encontrar en el menú de la figura 31, no obstante, en lugar de seleccionar "AOI-only" seleccionaremos "Fixed Point".

"Fixed Point" nos permitirá identificar un objeto y calcular sus coordenadas mediante la creación de espacios de trabajo. Profundizando lo previamente mencionado, esta función nos ayudará a identificar un objeto dentro de la visión del robot, calcular sus coordenadas y a la ayuda de otras funciones, desplazarnos hacia él sin importar dónde esté situado. Es esta función la que nos ayudará a trasladar los cilindros colocados de manera aleatoria en el segundo puesto de trabajo al primer puesto de trabajo.

Una vez seleccionado "Fixed Point", se nos pedirá escoger la calibración de referencia (figura 36) a la que en nuestro programa llamamos "bloques". El procedimiento que se ha de tener en cuenta una vez estemos dentro de dicha función es el mismo que el que fue empleado a la hora de programar "Pattern Matching (Shape)".

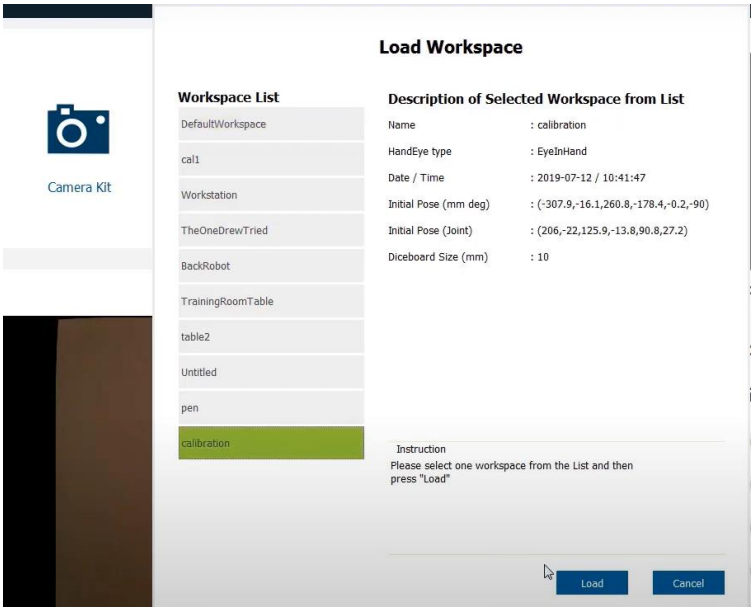

Figura 36: Pestaña de selección de calibración

Nota: la imagen de referencia para la figura 36 es una imagen de apoyo que no contiene las calibraciones aplicadas en este programa.

Tan pronto se haya culminado con la programación del nodo visión, se procederá a mover el brazo robótico hacia la pieza de la manera más precisa posible a la ayuda del botón de libre movimiento (figura 6). Una vez nos encontremos en la posición adecuada, para que la posición se guarde en el programa, hemos de presionar el botón "Point" localizado a la izquierda del botón de libre movimiento. Es importante notar que la base sobre la cual el punto tiene que ser guardada es la base creada por el nodo visión previamente programado. Esto último puede verificarse tanto en el símbolo que aparece a la derecha del punto (figura 37), como en la base actual de programación del robot (figura 12).

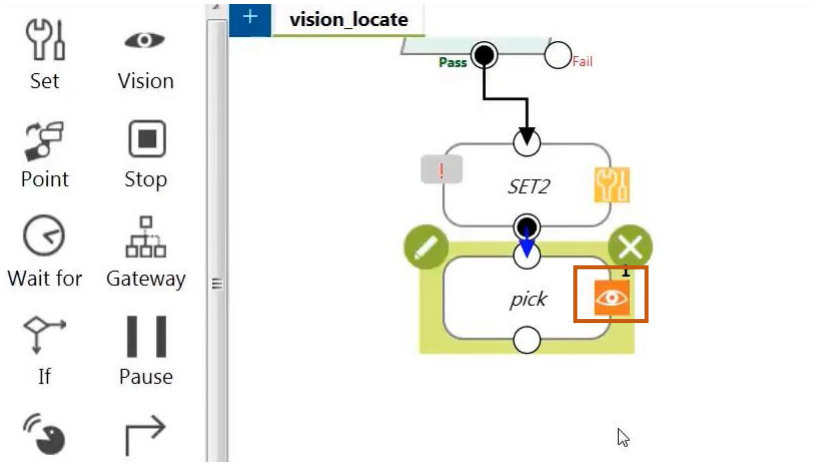

Figura 37: Base del punto de recogida

En cuanto hayamos terminado de interiorizar y programar las diferentes funciones claves de la

cámara, podremos comenzar con la programación del proyecto llamado tres en raya. Como mencionado previamente, el programa tiene como principales componentes esquina, centro y borde. Estas componentes descritas, grosso modo, en el diagrama de flujo del programa (figura 22) serán las que determinen la estrategia de juego a ser tomada para asegura la victoria o el empate.

3.2.2 Main

El "main" o página principal de programación será el punto de partida del programa. Generalmente, está destinado a controlar la ejecución del programa dirigiendo la dirección de ejecución hacia otras funciones del programa.

En el programa en cuestión, la primera página de programación estará compuesta por un número limitado de bloques, esto con la finalidad de volver más gráfico el proceso de ejecución del programa. Dada la complejidad en variedad de funciones, explicaremos la funcionalidad de cada una de ellas conforme se vayan desarrollando.

El juego a ser programado comienza una vez el usuario posiciona el primer bloque en la plantilla, en términos de calibración de la cámara podemos hablar del primer puesto de trabajo. Para poder lograr lo mencionado, hemos de guardar en la memoria de la cámara tanto el reconocimiento de la plantilla del tres en raya como los objetos a ser utilizados (figura 38). Dado que la explicación de cómo se realiza la programación de la cámara reside en el apartado anterior, nos enfocaremos en los siguientes pasos de la programación.

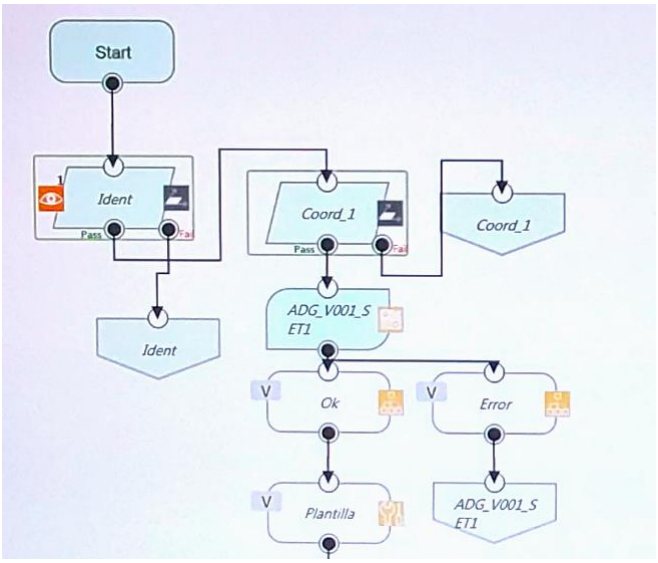

Figura 38: Inicio del main del primer programa

Tomando en consideración que la garra es un elemento externo al robot, es decir, un elemento que tiene la capacidad de ser removido del brazo robótico, debemos, antes de aplicar su funcionamiento, verificar su correcta conexión y ejecución sin ningún error. Para ello tendremos que utilizar el nodo de configuración de la garra (figura 40 y 41).

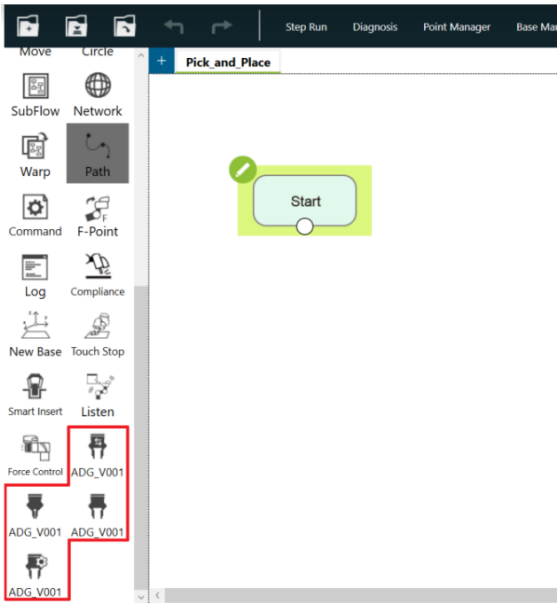

Figura 39: Nodos de uso de la garra

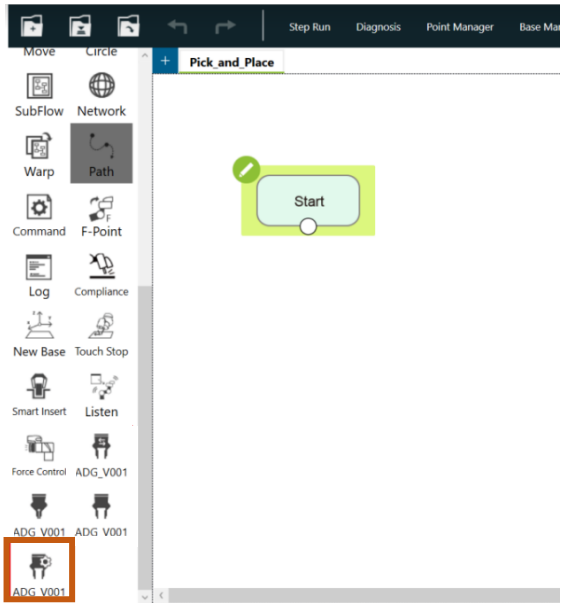

Figura 40: Nodo de configuración de la garra

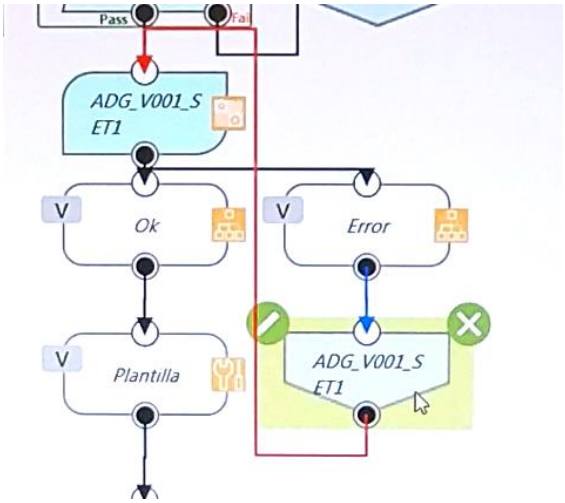

Figura 41: Ejemplo conexión nodo de configuración de la garra

Después de verificar el correcto funcionamiento de la garra, analizaremos la posición en la que se encuentra el bloque puesto por el usuario. A fin de encontrar la posición del bloque deberemos, por primera parte, tener en cuenta que las coordenadas devueltas por la cámara tendrán por origen de coordenadas la esquina de arriba izquierda de la foto tomada por la cámara. Por segunda parte, tendremos que definir rangos de distanciamiento pertinentes para establecer las posiciones en la plantilla.

El enfoque que tomaremos respecto a las coordenadas devueltas por la cámara se basará en calcular las distancias en los ejes X e Y tomando como referencia las coordenadas de otros elementos. Al usar este enfoque, podremos calcular la posición del primer bloque tomando como referencia las coordenadas de la plantilla (figura 42). De esta manera, comparando las distancias en los diferentes ejes junto con los rangos podremos establecer la posición del bloque. Clarificando el cálculo, definiremos las variables Dist_X=(Coord_1_N_X_TM-Ident_Plantilla_X_TM)*(-1) y

Dist_Y=(Coord_1_N_Y_TM-Ident_Plantilla_Y_TM)*(-1), donde la primera variable representa la coordenada del bloque, mientras que, la segunda representa la coordenada de la plantilla.

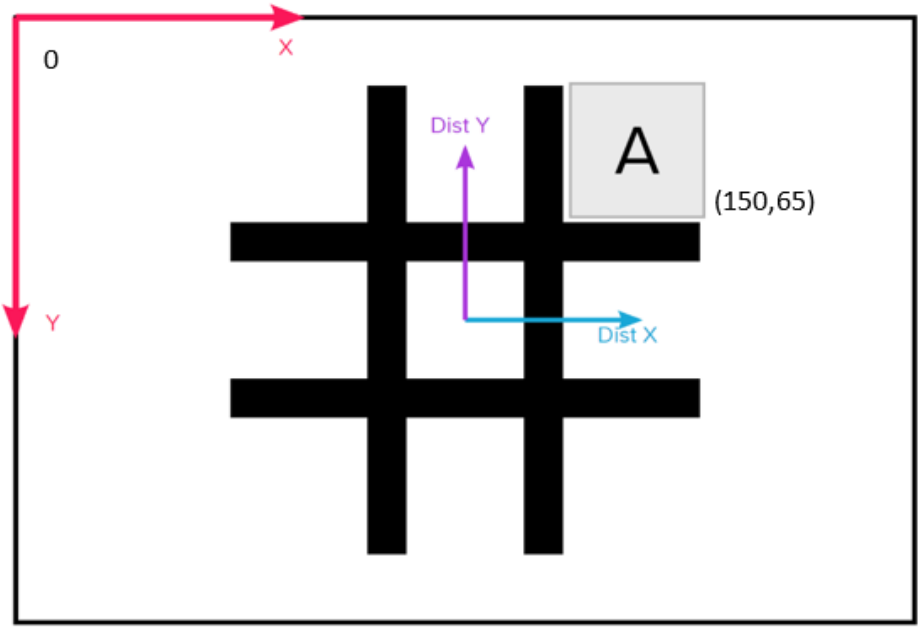

Figura 42: Origen de coordenadas y cálculo de las distancias

Teniendo en cuenta que las dimensiones de la plantilla son 22x12 cm, definimos lo siguiente:

- Si el bloque está alineado con el punto de referencia de manera horizontal, el valor absoluto de la distancia entre ambos en el eje Y deberá ser inferior o igual a 45mm.
- En caso de que el bloque esté alineado verticalmente con su punto de referencia, el valor absoluto de la distancia entre ambos en el eje X ha de ser inferior o igual a 65mm.
- Para considerar que el bloque está en la esquina opuesta al punto de referencia en el eje x, el valor absoluto de la distancia en el mismo eje ha de ser superior o igual a 150mm.
- Si el bloque se encuentra en la esquina opuesta al punto de referencia en el eje y, el valor absoluto de la distancia en el mismo eje deberá ser superior a 95mm.

Una vez definidos los rangos somos capaces de identificar la posición de la pieza en la plantilla. Para garantizar una correcta identificación de la posición de la pieza, crearemos una función que tendrá por nombre "Posición". Dentro de la función programaremos una serie de condiciones que determine si el bloque está en alguna esquina, el centro o algún borde. De esta manera, en caso de que se encuentre en alguna de las posiciones previamente descritas, guardaremos valores relacionados a las posiciones en la variable "Pos". Más específicamente, en caso de que el bloque se encuentre en el centro la variable "Pos" tomará como valor 0, si se encuentra en los bordes podrá tomar como valores 3 o 1 y en caso de que se encuentre en alguna esquina tomará como valor 2.

Tan pronto sea identificada la posición del bloque, el programa desembocará en el nodo llamado "GateWay". En dicho nodo podremos definir los tres casos posibles y posteriormente desarrollar la programación de cada una de las estrategias (figura 43).

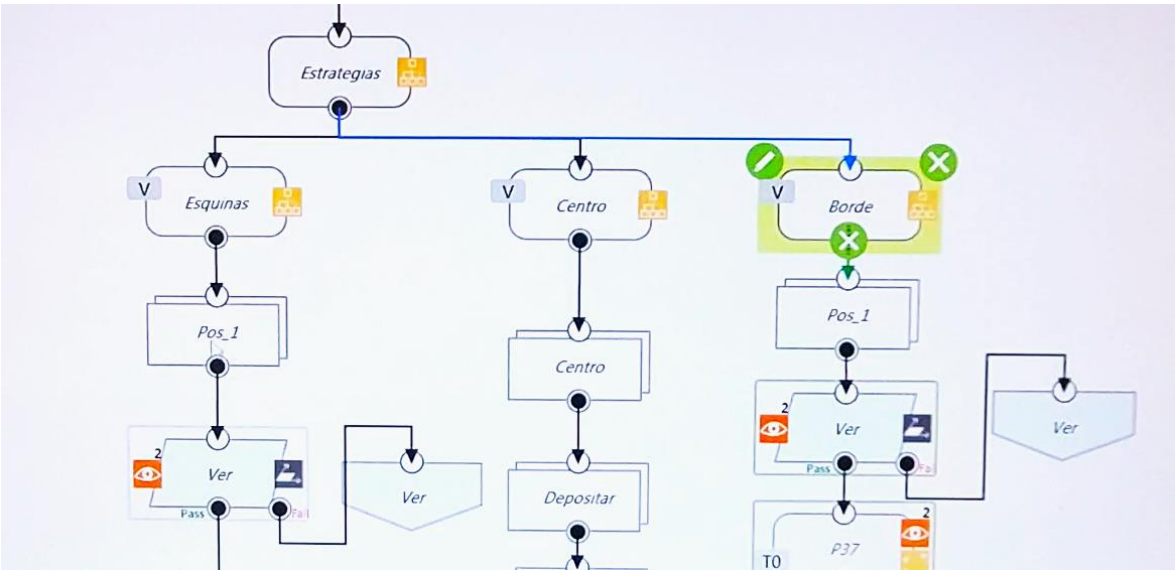

Figura 43: GateWay con las estrategias a ser desarrollada

En el main del programa encontraremos los inicios de las programaciones de las estrategias dependientes de la posición inicial del bloque. Sin embargo, explicaremos más detalladamente tanto el estudio como la programación de cada una de ellas en los siguientes apartados.

3.2.3 Esquina

En vista de que el objetivo de este programa es eliminar toda posibilidad de perder ante un usuario, es imperativo definir una estrategia infalible para cada una de las posibilidades. En este apartado nos enfocaremos en la estrategia necesaria una vez el usuario coloque el primer bloque en alguna de las esquinas de la plantilla.

Dado que nuestro robot siempre comenzará como segundo las probabilidades que tenemos de perder son mayores que nuestro contrincante. Tomando como referencia los datos proporcionados por la revista "American Mathematical Monthly" publicados en la revista de junio-julio 1958, las probabilidades que posee el primer jugador de tres en raya son de aproximadamente el 58.4%. Bajo el objetivo de lograr lo propuesto, tendremos que reducir el número de combinaciones posibles eliminando las combinaciones que guíen a un directo fracaso. Dichas combinaciones, sin tener en cuenta sus variaciones, pueden verse representadas en la figura 44.

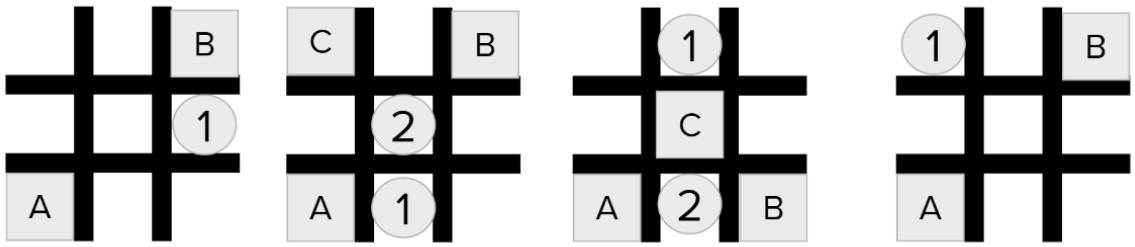

Figura 44: Ejemplos de combinaciones que llevan al fracaso

Recopilando las combinaciones a no ser consideradas, nos quedamos con una combinación inicial posible. Una vez se inicie el juego colocando el bloque en una de las esquinas el programa deberá colocar su primera pieza al centro de la plantilla, disminuyendo así la probabilidad de pérdida.

El traslado de los cilindros del segundo puesto de trabajo al primer puesto de trabajo se dividirá en dos partes. Una vez reconocido el cilindro en el segundo puesto de trabajo el robot deberá desplazarse al punto de calibración del primer puesto de trabajo. Una vez llegado este punto es importante realizar el mismo proceso de "Fixed Point" para lograr que el robot identifique la plantilla, sin importar su orientación, y se desplace al centro de la misma dejando una altura de aproximadamente 100mm. Este punto será el punto de partida de todos los movimientos para terminar de colocar los cilindros en la plantilla.

Examinando las posibilidades presentes una vez el bloque inicial se introduzca en una de las esquinas, podemos destacar la existencia de cuatro. Para poder determinar de la manera más sencilla y general los movimientos que el robot tenga que efectuar una vez se encuentre en una de estas posibilidades, concebimos la creación de las variables Camb_X, Camb_Y, Cam_x y Cam_y. Estas variables estarán programadas y definidas en la función llamada "Varia_Esquina", haciendo alusión a la variación que representa cada esquina de la plantilla en las variables. Entrando más en detalle, estas variables están directamente ligadas a la esquina en la cual se encuentra el bloque, esto puede verse retratado en la figura 45.

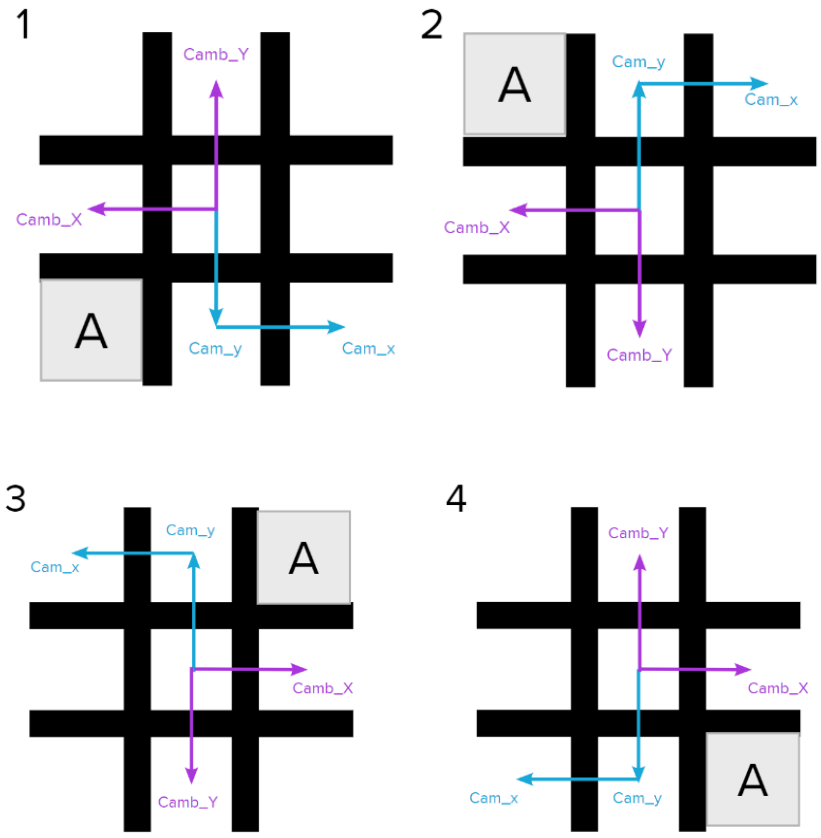

Figura 45: Variables en función de la esquina de inicio

Tan pronto sea reconocido el número de esquina que representa la situación actual, las variables adquirirán su valor, que en valor absoluto cumplirán Cam_y – Camb_Y=0 y Cam_x – Camb_X=0, siendo a su vez en valor absoluto Cam_x=100 y Cam_y=70.

Una vez se haya posicionado el cilindro en el centro, pasaremos a identificar el segundo bloque puesto por el usuario. La posición del segundo bloque dará nacimiento a siete funciones diferentes. Como ilustrado en el diagrama de flujo del programa (figura 22), cada una de las combinaciones da lugar a su propia función. Adicionalmente, es importante mencionar que las condiciones que definan la situación actual deberán ser diferentes para las esquinas impares y para las esquinas pares. Esto último, se debe a que el objeto de referencia para calcular las distancias junto con los rangos a ser considerados para las condicionantes cambiará entre esquinas pares e impares.

E_C_R

La primera función del programa será la que tenga las siglas "E_C_R" denominando Esquina Cerca Rosa. La programación de esta función, junto con las cuatro siguientes, serán las más sencillas. Considerando que para que esta combinación tome lugar el segundo bloque ha de estar en la misma vertical que el primer bloque y que la variables de movimiento están sujetas a las esquinas, la programación será general para todas las variaciones de esquina. Esto último es aplicable a las cuatro funciones siguientes.

A lo largo de la programación se intentará tener en cuenta el mayor número de combinaciones posibles. No obstante, esta combinación presenta uno de los menores números de combinaciones posibles. A continuación, la figura 46 mostrará las diferentes combinaciones y resultados una vez el juego se encuentre en esta situación.

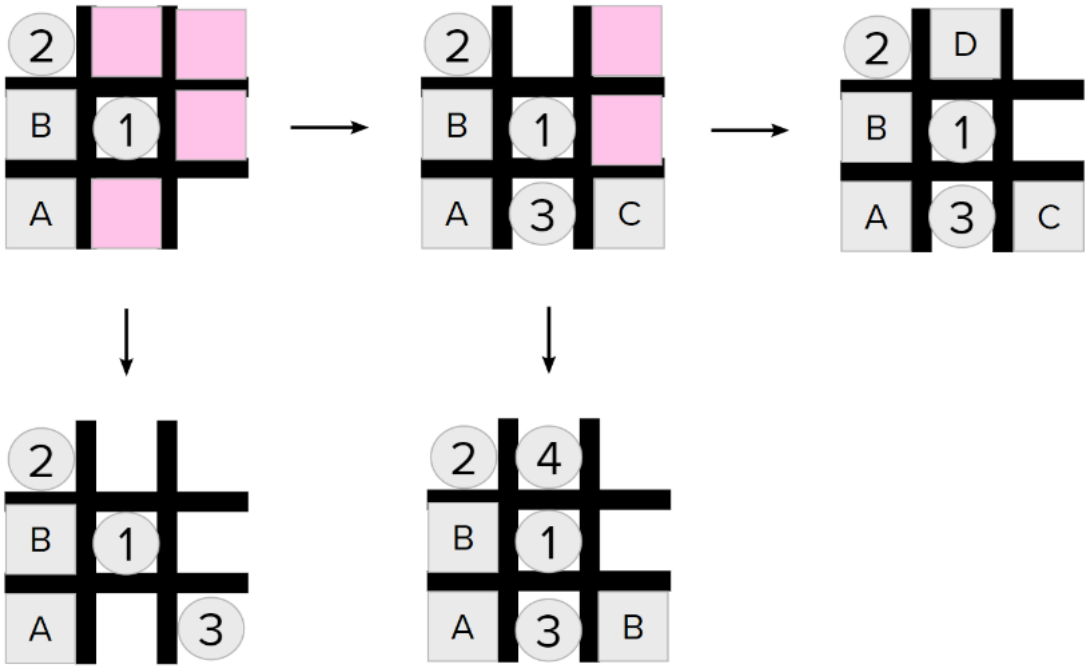

Figura 46: Combinaciones posibles E_C_R

Nota: es importante mencionar que el color rosado representa las posiciones en las cuales el usuario puede poner el siguiente bloque sin entorpecer nuestra victoria. Se mantendrá este código de color para las siguientes ilustraciones.

La programación paso a paso de esta función se realizará de la siguiente forma:

1- Una vez identificada la posición del segundo bloque, se procede a desplazar el cilindro sobre la plantilla usando las variables Camb_X y Camb_Y.
2- Identificamos si la posición del tercer bloque bloquea nuestra victoria.
    a. En caso de que bloquee nuestra victoria, el cilindro tendrá que bloquear la jugada del usuario usando la variable Cam_y.
        i. Si al identificar el cuarto bloque este nos bloquea, el juego termina en un automático empate.
        ii. Contrariamente, si el cuarto bloque no bloquea nuestra jugada, la variable a usar para el movimiento del cilindro será Camb_Y, marcando nuestra victoria.
    b. Si el bloque se encuentra en cualquiera de las áreas rosas del primer caso de la figura 46, las variables a ser usadas para el movimiento del cilindro son Cam_x y Cam_y, concluyendo con nuestra victoria.

E_C_V

La siguiente función que analizaremos será la que tenga las siglas "E_C_V" denominando Esquina Cerca Verde. La programación de esta función sigue, como explicado anteriormente, la misma línea de pensamiento de la función explicada anteriormente, pero con un grado más de . Por su parte, la figura 47 ilustrará las combinaciones posibles desatadas de estas primeras posiciones.

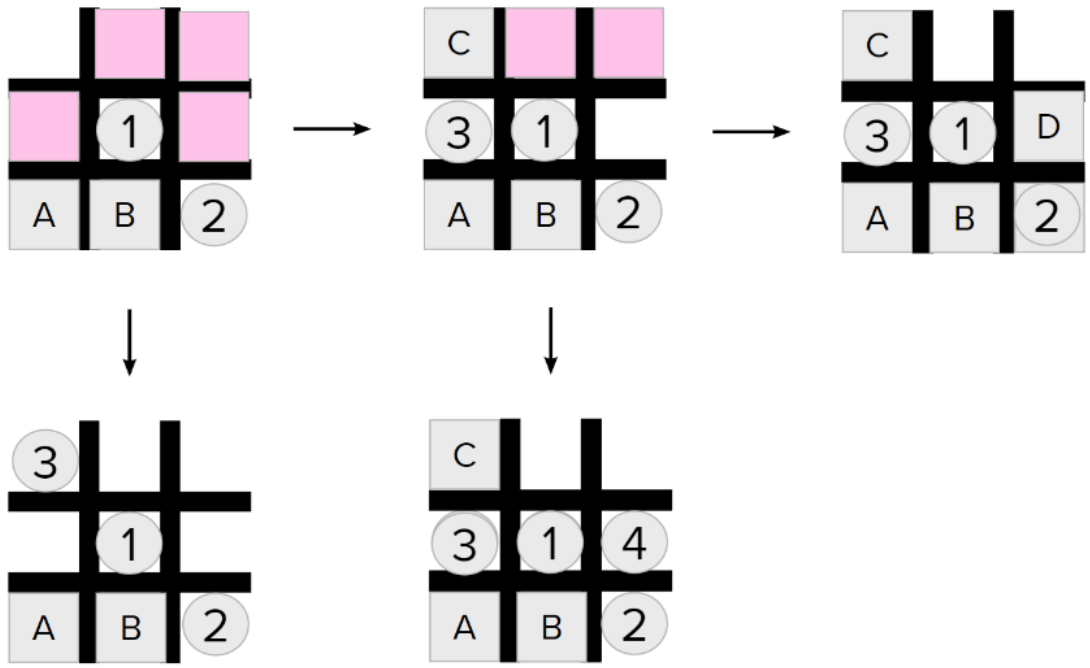

Figura 47: Combinaciones posibles E_C_V

Si bien Los pasos a seguir para programar exitosamente esta función serán los mismos que para la función anterior, se tendrán que realizar unos cambios consternando las variables. Sin mayor demora, los pasos a seguir serán los siguientes:

1- Una vez identificada la posición del segundo bloque, se procede a desplazar el cilindro sobre la plantilla usando las variables Cam_x y Cam_y.
2- Identificamos si la posición del tercer bloque bloquea nuestra victoria.
    a. En caso de que bloquee nuestra victoria, el cilindro tendrá que bloquear la jugada del usuario usando la variable Camb_X.
        i. Si al identificar el cuarto bloque este nos bloquea, el juego termina en un automático empate.
        ii. Contrariamente, si el cuarto bloque no bloquea nuestra jugada, la variable a usar para el movimiento del cilindro será Cam_x, marcando nuestra victoria.
    b. Si el bloque se encuentra en cualquiera de las áreas rosas del primer caso de la figura 47, las variables a ser usadas para el movimiento del cilindro son Camb_X y Camb_X, concluyendo con nuestra victoria.

E_L_R

"E_L_R" denominado como Esquina Lejos Verde será la programación que desarrollaremos a continuación. Como en las funciones anteriores, la programación de esta función será general a todas las esquinas, esto gracias a la función previamente programada " Varia_Esquina". La siguiente figura mostrará gráficamente las combinaciones posibles una vez nos encontremos en esta situación.

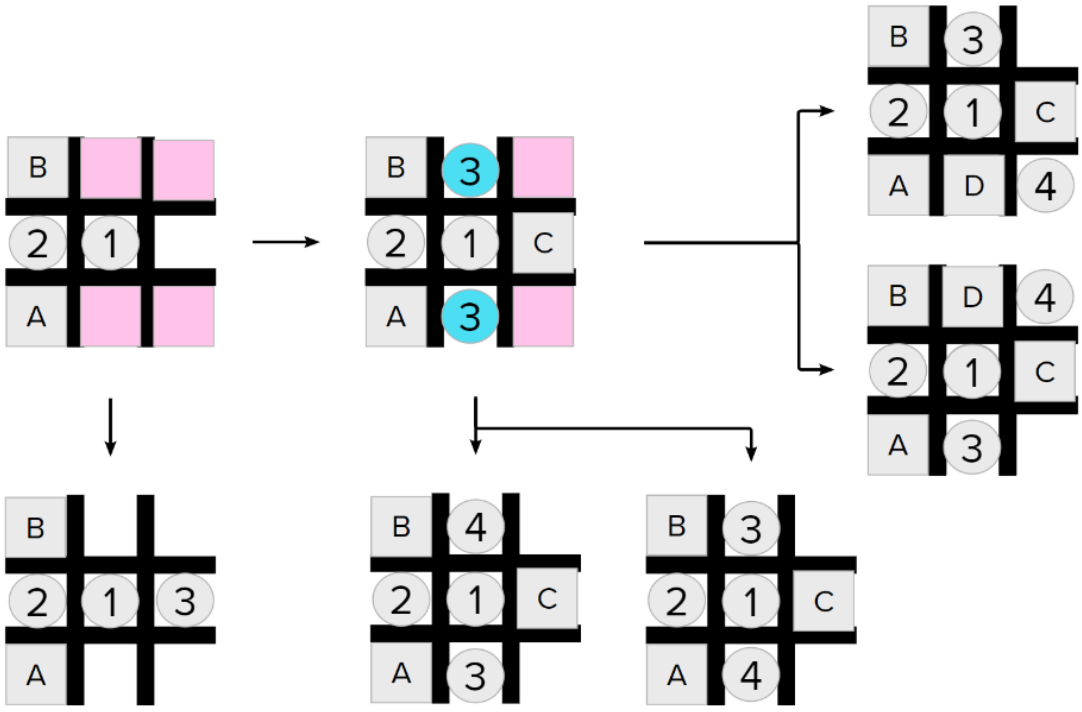

Figura 48: Combinaciones posibles E_L_R

Nota: los cilindros marcados de color celeste ilustran las dos posibilidades que se pueden tomar en cuenta para continuar el juego.

Si bien Los pasos a seguir para programar exitosamente esta función serán casi los mismos que para la función anterior, se tendrán que realizar unos cambios consternando tanto las variables como las nuevas posibilidades. Sin mayor demora, los pasos a seguir serán los siguientes:

1- Una vez identificada la posición del segundo bloque, se procede a desplazar el cilindro sobre la plantilla usando la variable Camb_X.
2- Identificamos si la posición del tercer bloque bloquea nuestra victoria.
    a. En caso de que bloquee nuestra victoria, el cilindro tendrá dos posibilidades de juego. A fin de poder programar ambas posibilidades, nos ayudaremos de la generación de una variable con una función aleatoria para que sea esta la que determine cuál de las dos debemos aplicar. Para simplificar la explicación, la primera posibilidad será el posicionamiento del cilindro gracias a la variable Camb_Y siendo la segunda posibilidad el posicionamiento del cilindro en base a la variable Cam_y.
        i. Si al identificar el cuarto bloque este nos bloquea, el juego terminará en un empate posicionando nuestro cuarto cilindro, en el primer caso, en base a las variables Cam_x y Cam_y. En el segundo caso, deberemos posicionarlo en base a Cam_x y Camb_Y.
        ii. Contrariamente, si el cuarto bloque no bloquea nuestra jugada, la variable a usar para el movimiento del cilindro será Cam_y en el primer caso y Camb_Y en el segundo, marcando nuestra victoria.
    b. Si el bloque se encuentra en cualquiera de las áreas rosas del primer caso de la figura 47, la variable a ser usada para el movimiento del cilindro es Cam_x, concluyendo con nuestra victoria.

E_L_V

La cuarta función por programar será "E_L_V" que denota Esquina Lejos Verde. Esta función se desenvolverá de la misma manera que la función anterior teniendo en cuenta ciertas variaciones en las variables de movimiento. Como ilustrado en las funciones anteriores, presentaremos las combinaciones posibles en la figura 49.

La programación de esta función se llevará a cabo de la siguiente forma:

1- Una vez identificada la posición del segundo bloque, se procede a desplazar el cilindro sobre la plantilla usando la variable Cam_y.
2- Identificamos si la posición del tercer bloque bloquea nuestra victoria.
    a. En caso de que bloquee nuestra victoria, el cilindro tendrá dos posibilidades de juego. A fin de poder programar ambas posibilidades, nos ayudaremos de la generación de una variable con una función aleatoria para que sea esta la que determine cuál de las dos debemos aplicar. Para simplificar la explicación, la primera posibilidad será el posicionamiento del cilindro gracias a la variable Camb_X siendo la segunda posibilidad el posicionamiento del cilindro en base a la variable Cam_x.

i. Si al identificar el cuarto bloque este nos bloquea, el juego terminará en un empate posicionando nuestro cuarto cilindro, en el primer caso, en base a las variables Cam_x y Camb_Y. En el segundo caso, deberemos posicionarlo en base a Camb_X y Camb_Y.
ii. Contrariamente, si el cuarto bloque no bloquea nuestra jugada, la variable a usar para el movimiento del cilindro será Cam_x en el primer caso y Camb_X en el segundo, marcando nuestra victoria.

b. Si el bloque se encuentra en cualquiera de las áreas rosas del primer caso de la figura 47, la variable a ser usada para el movimiento del cilindro es Camb_Y, concluyendo con nuestra victoria.

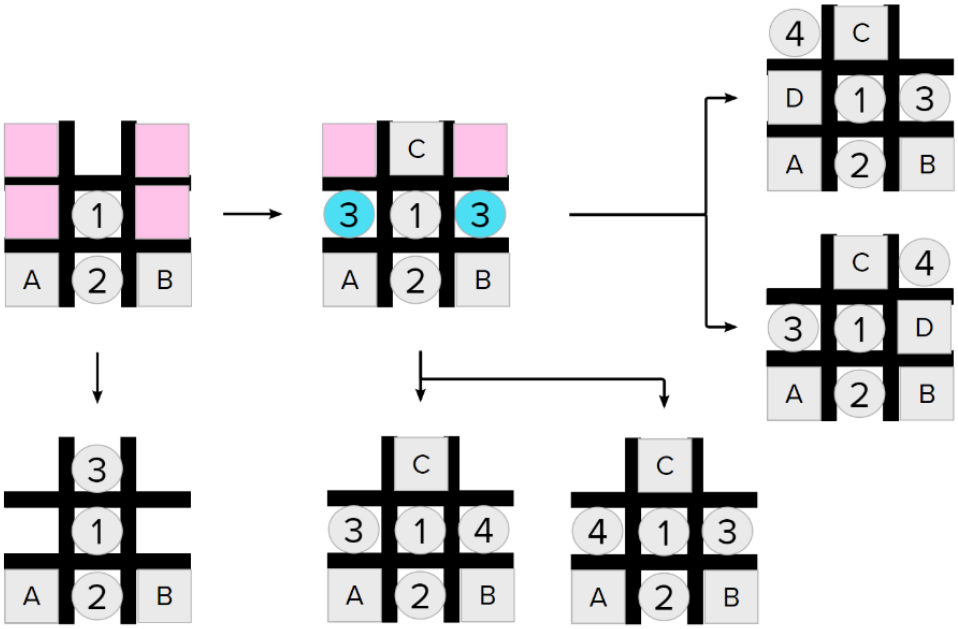

Figura 49: Combinaciones posibles E_L_V

E_A

Las siglas que determinan esta función denotan Esquina Arriba, haciendo alusión a la posición centrada superior. La programación de esta función será una de las más difíciles en esta sección del programa. Su dificultad reside en la cantidad de combinaciones posibles. A continuación, se mostrarán tanto las diferentes combinaciones y posiciones que tomaremos en consideración (figura 50), como las posiciones que no tendremos en cuenta dada su ineficiencia (figura 51).

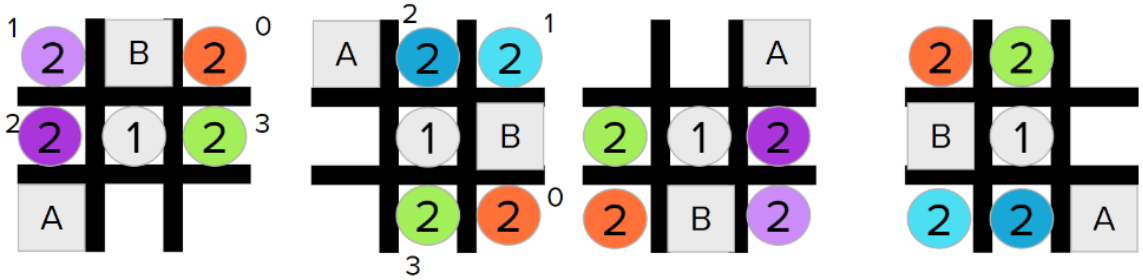

Figura 50: Posiciones posibles E_A

Nota: observando la anterior figura, podemos destacar la presencia de numeraciones a las posiciones de los cilindros. Dado que se presentan varias opciones de posicionamiento del cilindro, utilizaremos una función que genere números aleatorios. Las numeraciones presentes al costado de las posiciones representan el número al cuál estas pueden ser atribuidas.

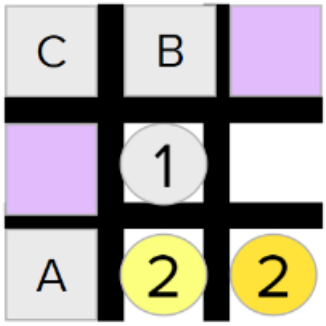

Figura 51: Posiciones no consideradas E_A

Nota: los diferentes colores del segundo cilindro muestran las dos posibilidades que quedan a ser tomadas en cuenta. Los cuadrados morados indican las posiciones que nuestro tercer cilindro puede tomar, no obstante, esto no impedirá la inmediata victoria del usuario.

Teniendo en cuenta que tenemos cuatro posiciones posibles para del segundo cilindro, desarrollaremos la programación y combinaciones de cada una de ellas. A fin de organizar la explicación, comenzaremos en orden creciente de los valores de la variable aleatoria. Adicionalmente, como podemos observar en la figura 50, la programación de esta función tendrá que dividirse en dos partes. La primera parte, será la programación de en caso de las esquinas 1 y 3, pigmentación morada de los segundos cilindros (figura 50). La segunda parte, estará marcada por la programación en caso de las esquinas 2 y 4, pigmentación azulada del segundo cilindro (figura 50).

El primer caso que tendremos en cuenta será la programación de la posibilidad cuyo valor de la variable aleatoria sea nulo. A dicho caso nos referiremos como “diagonal”. Bajo el objetivo de programar la diagonal, comenzaremos ilustrando los diferentes casos posibles tanto para las esquinas impares (figuras 52-56) como para las pares (figuras 57-61).

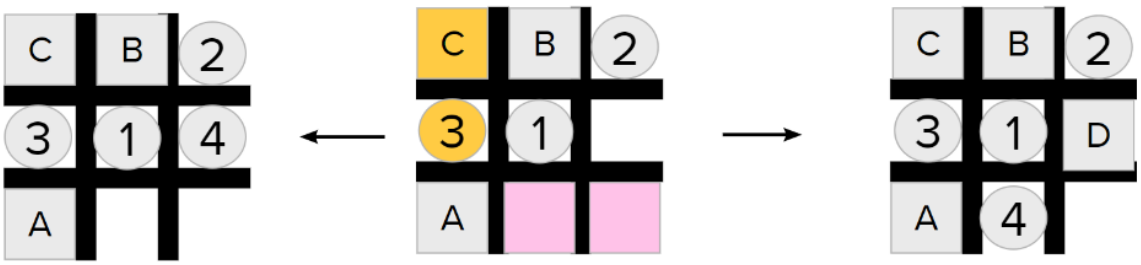

Figura 52: Primera posición esquina impar E_A

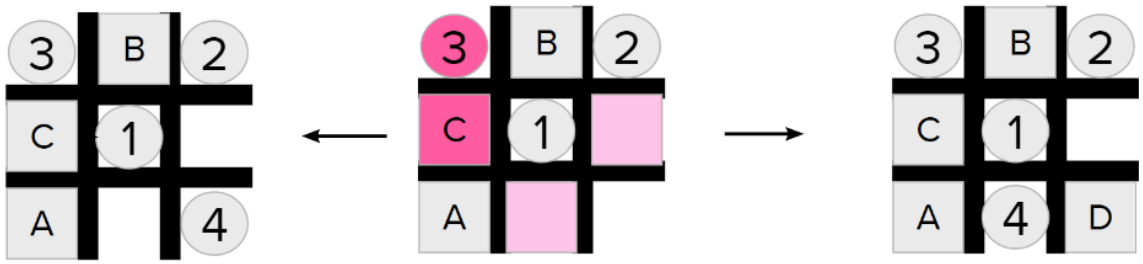

Figura 53: Segunda posición esquina impar E_A

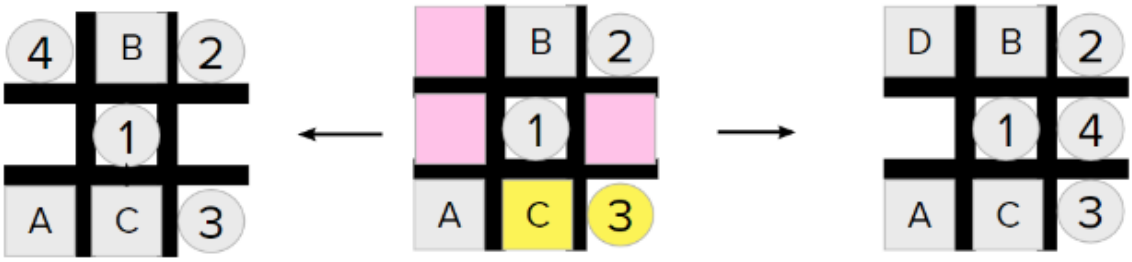

Figura 54: Tercera posición esquina impar E_A

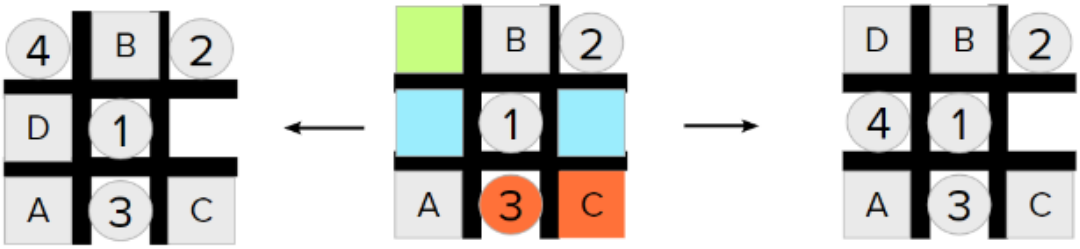

Figura 55: Cuarta posición esquina impar E_A

Nota: es importante recalcar el significado de los diferentes colores presentados en la anterior figura. El color celeste indica que, en caso de que el cuarto bloque se encuentre en alguna de las posiciones marcadas por este color, el cilindro tendrá que ubicarse en la posición marcada por el color verde y viceversa.

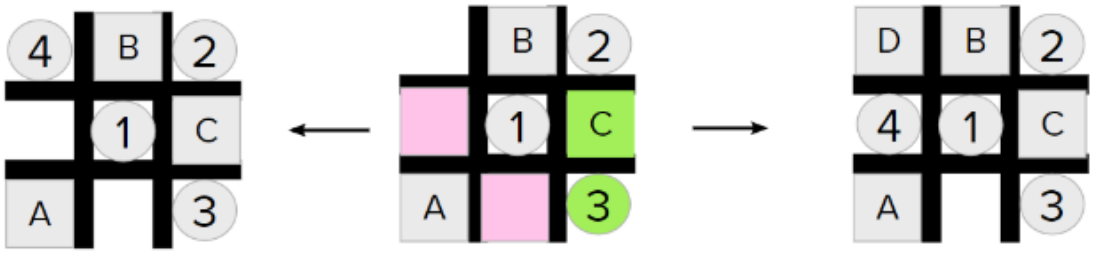

Figura 56: Quinta posición esquina impar E_A

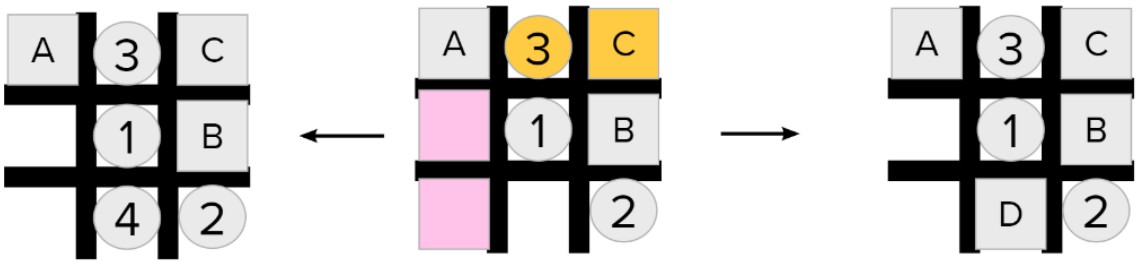

Figura 57: Primera posición esquina par E_A

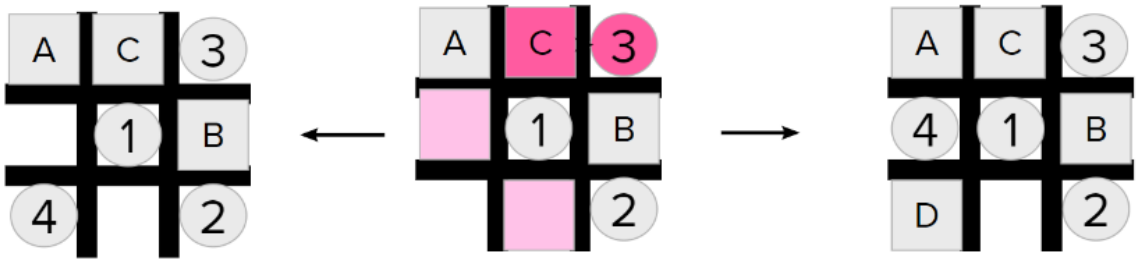

Figura 58: Segunda posición esquina par E_A

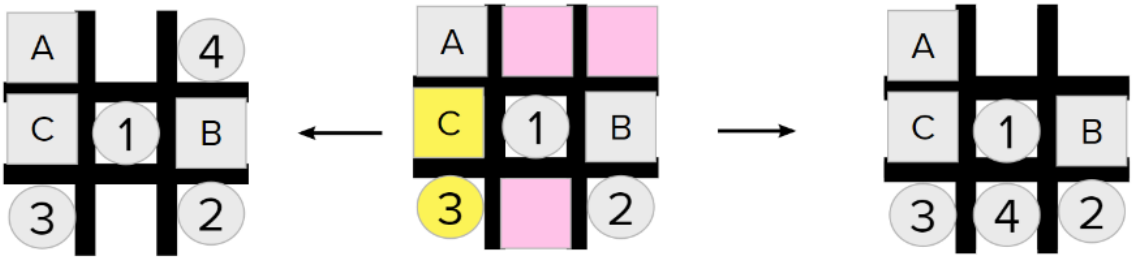

Figura 59: Tercera posición esquina par E_A

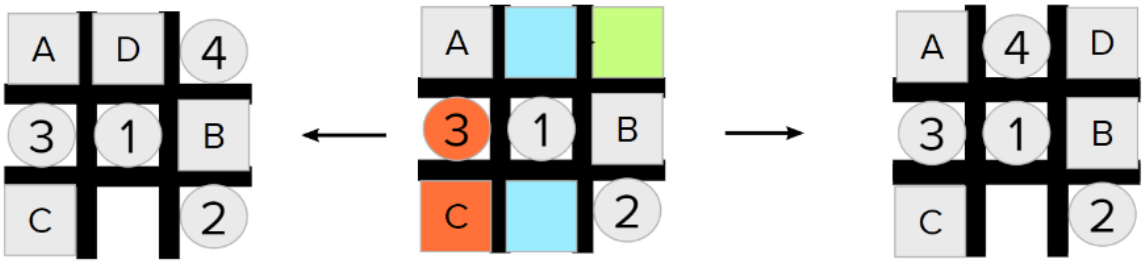

Figura 60: Cuarta posición esquina par E_A

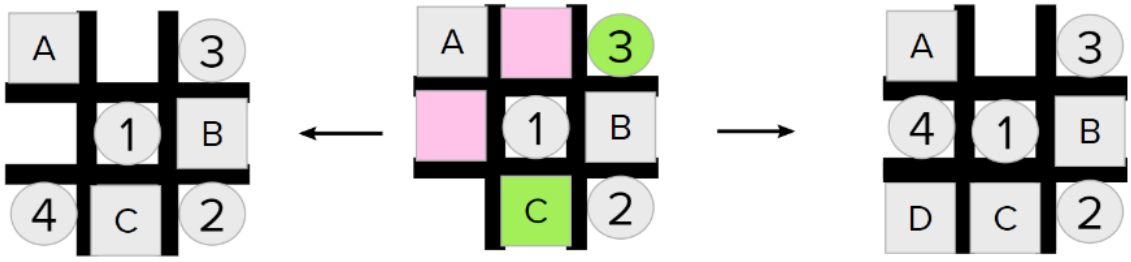

Figura 61: Quinta posición esquina par E_A

Los pasos por seguir para programar las combinaciones previamente mostradas son prácticamente

los mismos entre ellos. A grandes rasgos, la línea de pensamiento que tendremos que seguir es la siguiente:

1- Identificar la posición del tercer bloque puesto por el usuario. Para lograr lo mencionado podremos cambiar, dependiendo del caso en cuestión, el objeto de referencia de medición de distancias.
2- Una vez identificada la posición, comparar la esquina de inicio junto con la posición a los casos previamente presentados.
3- Tan pronto se identifique un caso, desplazar el tercer cilindro a la posición mostrada por la figura representante del caso.
4- Identificar la posición del cuarto bloque, en caso de que sea necesario variar el objeto de referencia de medición de distancias.
5- Una vez instaurada la condición, posicionar el cuarto cilindro de la manera que más nos convenga.

Bajo la idea de ilustrar los pasos previamente explicados, daremos un ejemplo de programación de uno de los casos precedentes. El caso que servirá como ejemplar será el representado por la figura 59. Para identificar la posición del tercer bloque tomaremos como objeto de referencia de medición de distancias las coordenadas en el eje Y de la plantilla. Si se presenta el caso en el cual la distancia Dist_Y es inferior o igual a 45mm, podemos decir que el tercer bloque se encuentra en la posición ilustrada por la figura 59. Dado que no podemos permitir que el usuario gane, hemos de trasladar el tercer cilindro sobre la plantilla utilizando las variables Camb_Y y Camb_Y. Dado que sin importar la posición en la que se encuentre el siguiente bloque ganamos, podemos programar el posicionamiento del cuarto cilindro de dos maneras diferentes. En un primer enfoque, podremos tomar como referencia de medición el eje x de la plantilla. Si la distancia Dist_X es inferior o igual a 65mm quiere decir que el bloque se encuentra en la misma vertical que el centro de la plantilla. Si este es el caso, podremos mover el cuarto cilindro a la ayuda de las variables Cam_x y Cam_y, de lo contrario, moveremos el cilindro a la ayuda de la variable Camb_Y. En un segundo enfoque, podremos tomar como referencia las coordenadas del primer bloque. Al tomar estas coordenadas podremos verificar si el primer y cuarto bloque se encuentran en la misma horizontal, esto sólo si Dist_Y es inferior o igual a 45mm. En caso de que el cuarto bloque se encuentre en la misma horizontal, deberemos posicionar el cuarto cilindro a la ayuda de la variable Camb_Y, en caso contrario, tomaremos la ayuda de las variables Cam_x y Cam_y.

Como segundo y tercer caso de la programación, tomaremos los valores de la variable aleatoria que sea igual a uno y a dos respectivamente. Gratamente, tanto las combinaciones posibles como la programación de ambas serán sencillas. A continuación, mostraremos las combinaciones dadas de ambos casos separando las esquinas pares de las impares.

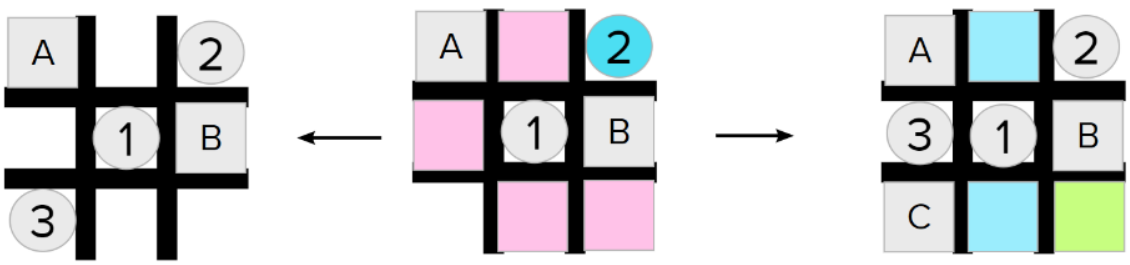

Figura 62: Segundo caso de programación esquina par E_A

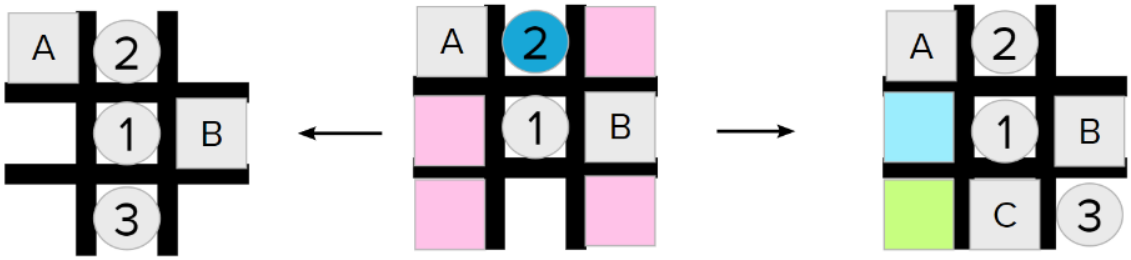

Figura 63: Tercer caso de programación esquina par E_A

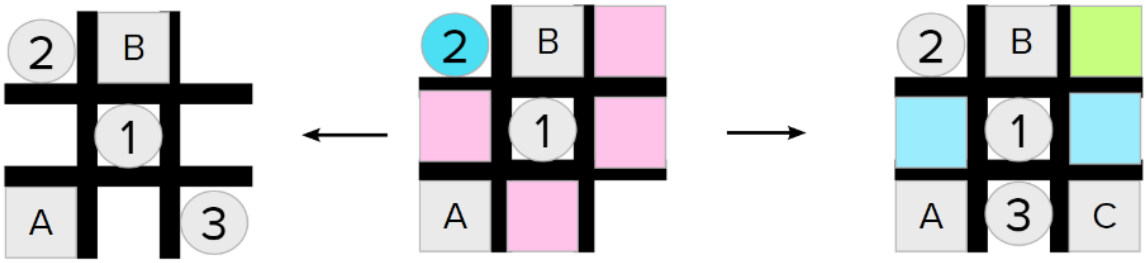

Figura 64: Segundo caso de programación esquina impar E_A

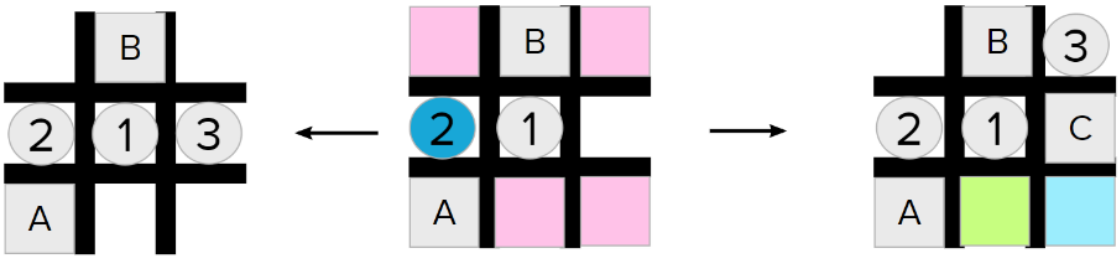

Figura 65: Tercer caso de programación esquina impar E_A

Usando como referencia de programación los pasos previamente descritos, desarrollaremos a detalle la programación de la figura 65. Antes de identificar la posición del tercer bloque tenemos que desplazar el segundo cilindro. Para esto último, usaremos las variables Camb_X y Camb_y. A fin de identificar la posición del tercer bloque de la manera más sencilla, usaremos como objeto de referencia la plantilla del tres en raya. En caso de que la distancia en el eje Y, Dist_Y, sea inferior o igual a 45mm el bloque introducido por el usuario bloqueará nuestro prospecto de victoria. En caso contrario, nuestra victoria se verá asegurada por el desplazamiento del tercer cilindro en base a la variable Cam_x. Siguiendo el caso de bloqueo, deberemos identificar la posición del cuarto bloque. Bajo este propósito, podremos tomar como referencia dos objetos, por primera parte, podremos tomar como referencia la plantilla. Tomando la plantilla como referencia, estableceremos que si la distancia en las coordenadas del eje X es inferior o igual a 65mm podremos asumir que el bloque se encuentra en el recuadro pintado con verde (figura 65). En este caso, trasladaremos el cuarto cilindro a la posición celeste gracias a las variables Cam_x y Cam_y. Por segunda parte, podremos tomar como referencia la posición del tercer bloque. De esta manera, si la variable Dist_X es inferior a 65mm el bloque se encontrará ubicado en el recuadro celeste (figura 65), por lo cual, el cilindro tendrá que ser trasladado al recuadro verde usando la variable Cam_y.

Finalmente, como cuarto caso de programación tendremos el valor de la variable aleatoria que sea igual a tres. A dicho caso nos referiremos como “medio derecha”. Para proseguir con la programación, mostraremos gráficamente las diferentes combinaciones posibles una vez se haya tomado este camino, dividiendo los casos de las esquinas impares y pares.

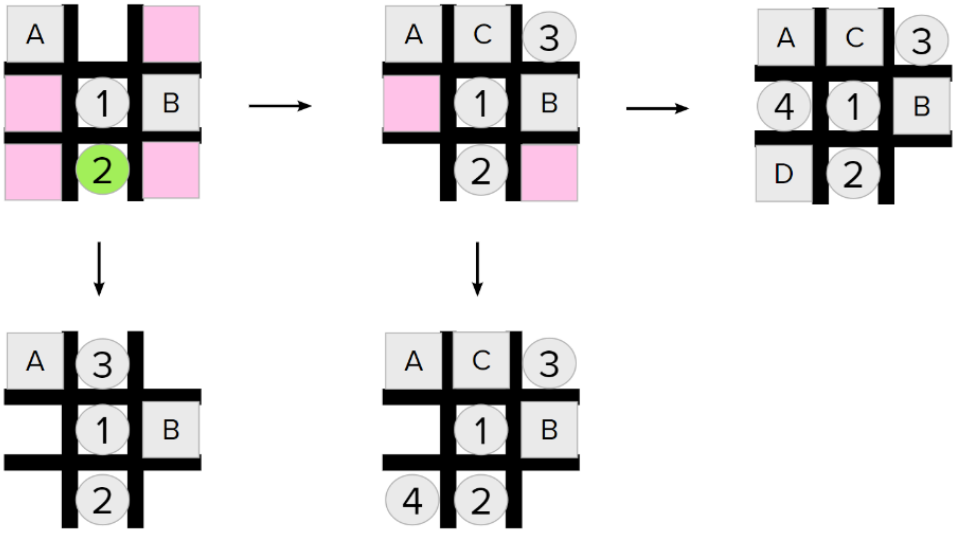

Figura 66: Cuarto caso de programación esquina par E_A

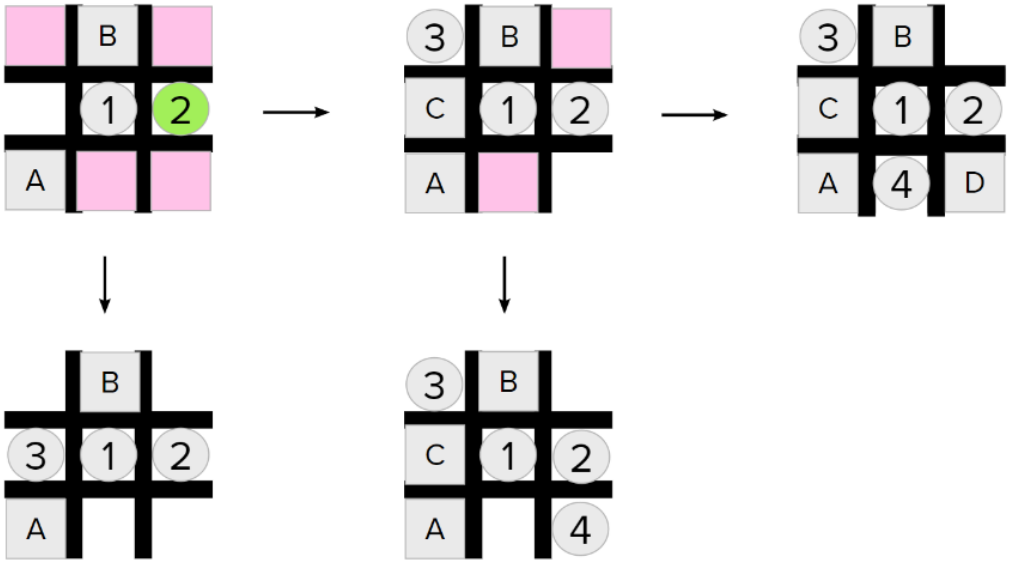

Figura 67: Cuarto caso de programación esquina impar E_A

Teniendo en mente la lógica de programación ya aplicada en las funciones y casos pasados podemos exponer minuciosamente la programación del caso ilustrado en la figura 67. Para comenzar la programación tenemos que trasladar el segundo cilindro a la posición marcada gracias a la variable Cam_x. Para identificar la posición del tercer bloque, usaremos como referencia las coordenadas de la plantilla. Por primera parte, suponiendo que el bloque se encuentre en la misma horizontal que el centro de la plantilla, Dist_Y inferior o igual a 45mm, el usuario nos bloquearía. Si esto se da, tendremos que bloquear el siguiente movimiento del usuario desplazando el cilindro en base a las variables Camb_X y Camb_Y. En caso de que el usuario nos vuelva a bloquear, posicionaremos el cuarto cilindro usando la variable Cam_y. En caso contrario al anterior, ganaremos empleando las variables Cam_x y Cam_y. Por segunda parte, teniendo en cuenta que el usuario no nos bloqueó con el tercer bloque, al aplicar la variable Camb_X quedaremos victoriosos.

E_Diag

Esta función fue llamada "E_Diag" ya que busca ilustrar en su nombre el caso en el que el usuario introduzca el segundo bloque en la esquina opuesta, formando así, una diagonal entre el primer y segundo bloque. Cabe a recalcar que una vez esta combinación se realice, la única manera de prevenir la derrota es colocando el segundo cilindro en alguno de los bordes de la plantilla. Para poder ilustrar la programación de esta función nos apoyaremos en las combinaciones posibles mostradas a continuación (figuras 68 a 71).

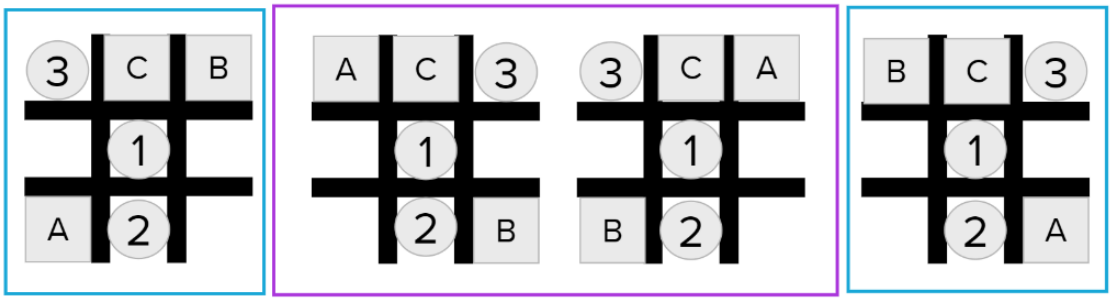

Figura 68: Caso segundo cilindro en el primer borde

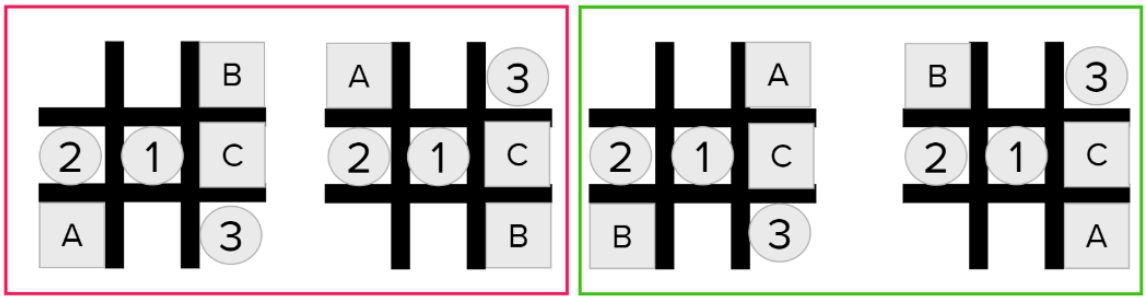

Figura 69: Caso segundo cilindro en el segundo borde

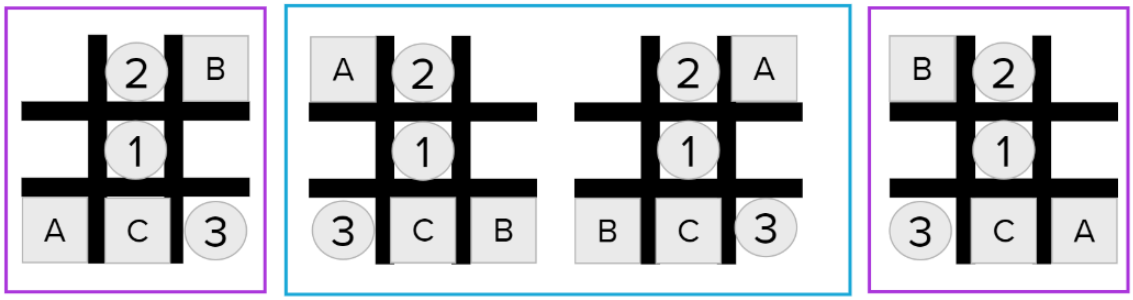

Figura 70: Caso segundo cilindro en el tercer borde

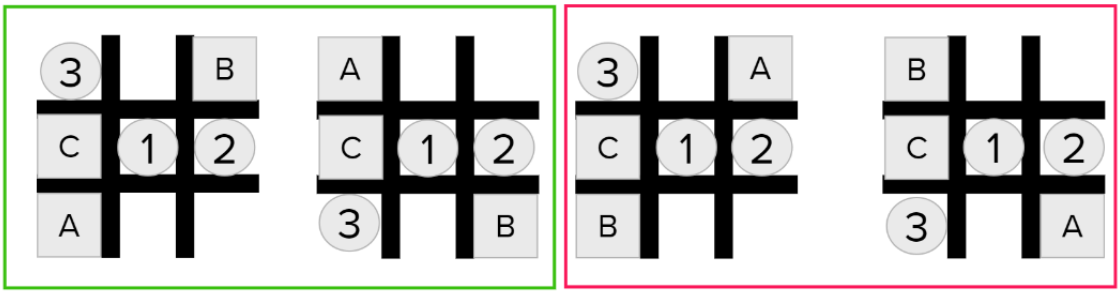

Figura 71: Caso segundo cilindro en el cuarto borde

Como mencionado previamente, la única manera de evitar la derrota será colocando el segundo cilindro en alguno de los bordes. A fin de mantener el máximo número de combinaciones posibles, tendremos en cuenta las cuatro combinaciones posibles por esquina. Al observar las figuras ilustradoras de las combinaciones posibles (figuras 68-71), podemos destacar los marcos de diferentes colores. Estos marcos determinarán las combinaciones similares y las programaciones que se lleven en paralelo.

A fin de aclarar la línea de pensamiento de la programación que se pretende llevar a cabo, explicaremos de manera detallada los pasos a seguir para la programación de la figura 71. Para conocer más profundamente las posibilidades de la figura en cuestión, presentaremos las combinaciones posibles de dos de las esquinas presentadas.

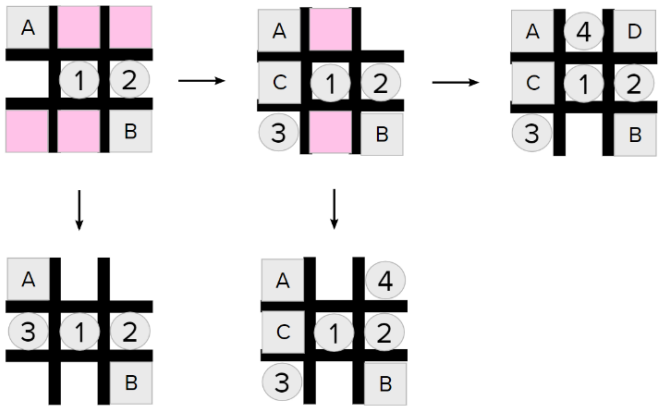

Figura 72: Caso segunda esquina en cuarta configuración

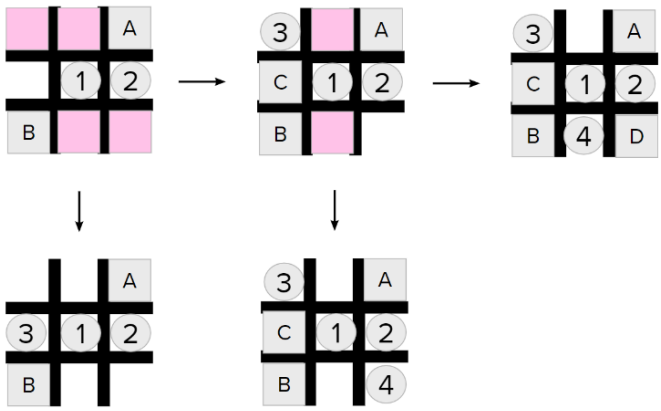

Figura 73: Caso tercera esquina en cuarta configuración

El primer paso para tener en cuenta es la igualdad en términos de identificación del tercer bloque. Para que el tercer bloque bloquee nuestra jugada, este ha de estar en la misma horizontal que el centro de la plantilla. A fin de verificar que esto se cumple, la variable Dist_Y tendrá como componentes tanto las coordenadas del bloque como las de la plantilla. Como explicado anteriormente, para considerar que el bloque está en la misma horizontal que la plantilla, el valor de Dist_Y tiene que ser inferior o igual a 45mm. Una vez hayamos tomado este paso en cuenta, pasaremos a dividir ambos casos. Más específicamente, por una parte, tendremos las esquinas 1 y 2, mientras que por el otro lado tendremos las esquinas 3 y 4.

Comenzando con la programación de la segunda esquina, una vez el tercer bloque se encuentre en una posición diferente a la que bloquearía nuestra jugada, tendremos que desplazar nuestro tercer cilindro hacia la victoria utilizando la variable Camb_X. En caso contrario, tendremos que bloquear, por nuestra parte, el siguiente movimiento del usuario. Para ello, moveremos el tercer cilindro bajo la influencia de las variables Camb_X y Camb_Y. A fin de verificar que el cuarto bloque no se encuentra entorpeciendo nuestro movimiento, hemos de tomar como coordenadas de referencia para el cálculo de las distancias las coordenadas del eje X de la plantilla. Una vez hayamos verificado que la distancia Dist_X es inferior a 65mm podremos proclamarnos como vencedores colocando nuestro cuarto cilindro en la posición definida por las variables Cam_x y Cam_y. En caso de que la variable Dist_X no cumpla con lo previamente mencionado, colocaremos el cuarto cilindro en la posición que marque la variable Cam_y.

Siguiendo con la programación de la tercera esquina, una vez verificada que la posición del tercer cilindro no nos bloquea la jugada, moveremos el cilindro a la ayuda de la variable Cam_x. En caso contrario, el cilindro se desplazará tomando en cuenta las variables Camb_X y Camb_Y. Si el cuarto cilindro se encuentra bloqueando nuestra victoria, tomando como referencia las coordenadas en X del primer bloque, hemos de colocar el cilindro en base a Camb_Y. Contrariamente, si el cuarto bloque no estropea nuestra victoria, moveremos el cilindro gracias a Camb_Y y Camb_X.

E_C

Este código representa el último grupo de combinaciones posibles denominado como Esquina Costado. La programación de esta sección se llevará a cabo de la misma manera que la programación de E_A. Primero, mostraremos las posiciones del segundo cilindro que tendremos en cuenta (figura 73). En segundo lugar, ilustraremos las diferentes combinaciones posibles que cada uno de estos casos teniendo en cuenta, por un lado, las esquinas pares y por otro las impares.

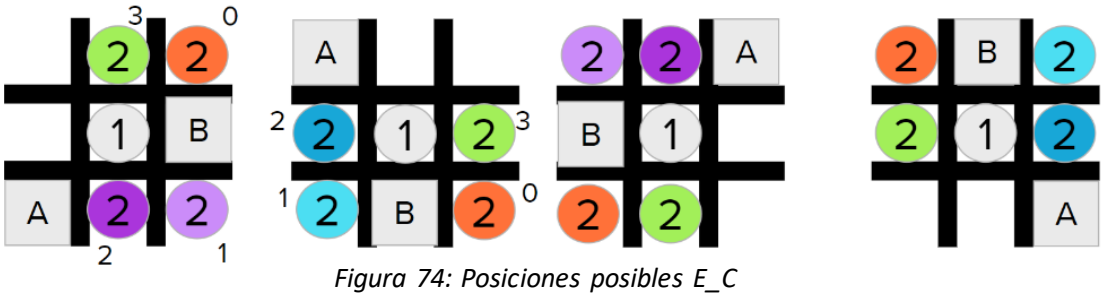

Figura 74: Posiciones posibles E_C

Como mencionado en la programación E_A, al tener cuatro posiciones posibles para el segundo cilindro, programaremos una variable de valor aleatorio del cero a tres para desarrollar la programación de todas las combinaciones posibles que surjan de dichas posiciones (figura 74).

Adicionalmente, tendremos en cuenta, como en programaciones anteriores, los valores de las esquinas. Los valores pares por un lado e impares por otro.

El primer caso de programación que explicaremos será el cuál se llevará a cabo una vez el valor de la variable aleatoria sea nulo. Más específicamente, en el caso en el que el segundo cilindro se encuentre en la diagonal opuesta al primer bloque. A continuación, veremos las diferentes combinaciones posibles y sus resultados divididos por las esquinas pares (80-84) e impares (75-79).

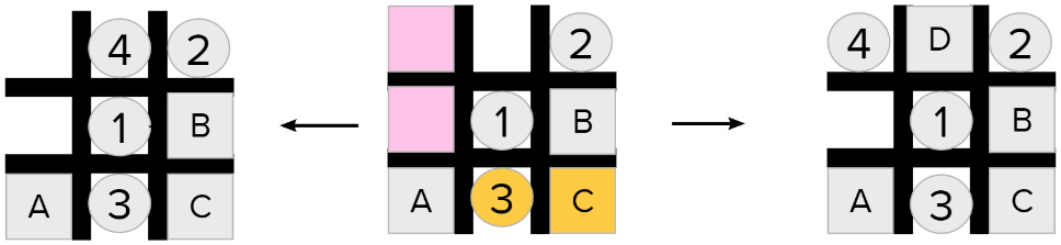

Figura 75: Primera posición esquina impar E_C

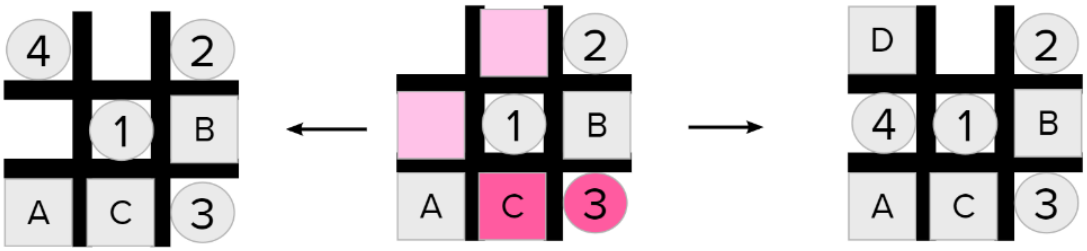

Figura 76: Segunda posición esquina impar E_C

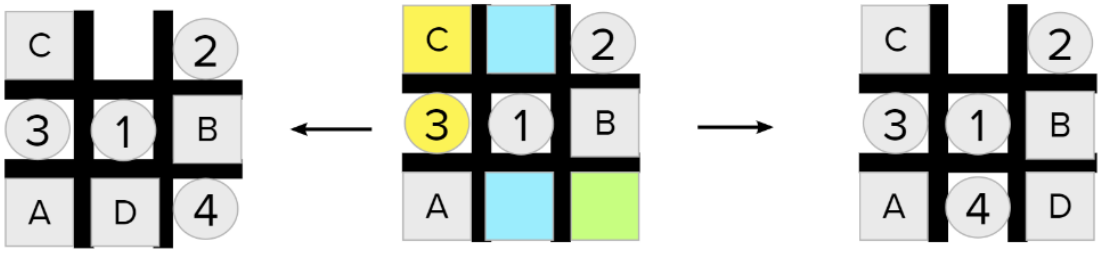

Figura 77: Tercera posición esquina impar E_C

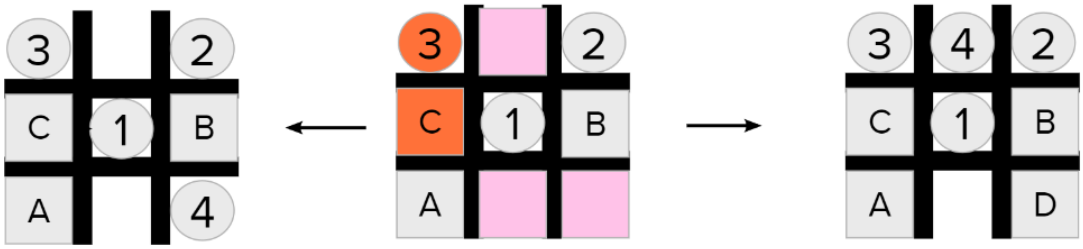

Figura 78: Cuarta posición esquina impar E_C

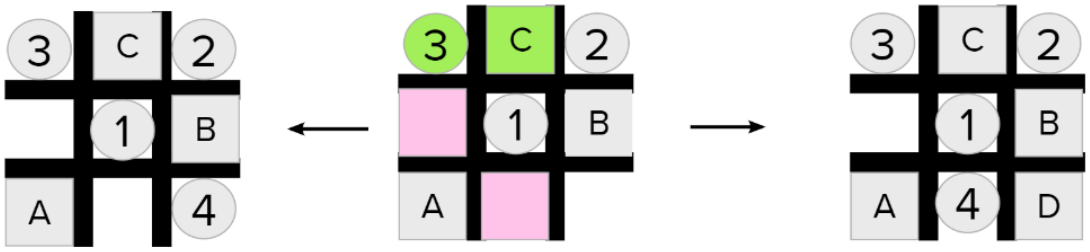

Figura 79: Quinta posición esquina impar E_C

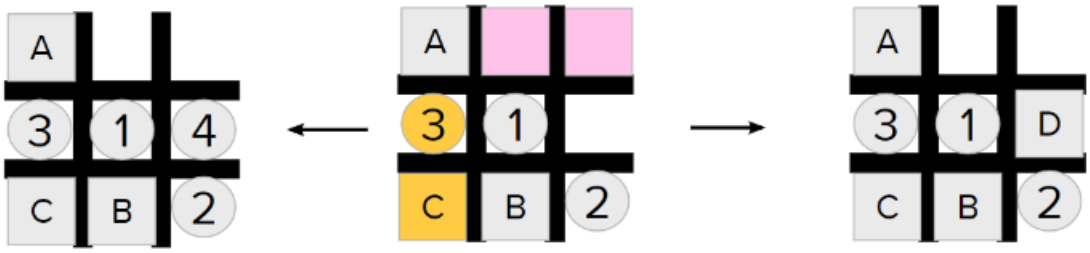

Figura 80: Primera posición esquina par E_C

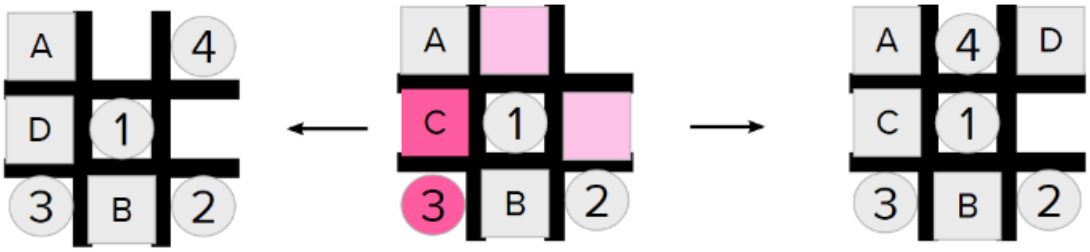

Figura 81: Segunda posición esquina par E_C

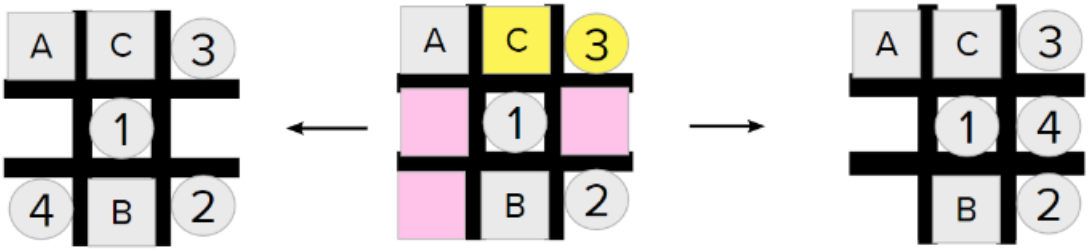

Figura 82: Tercera posición esquina par E_C

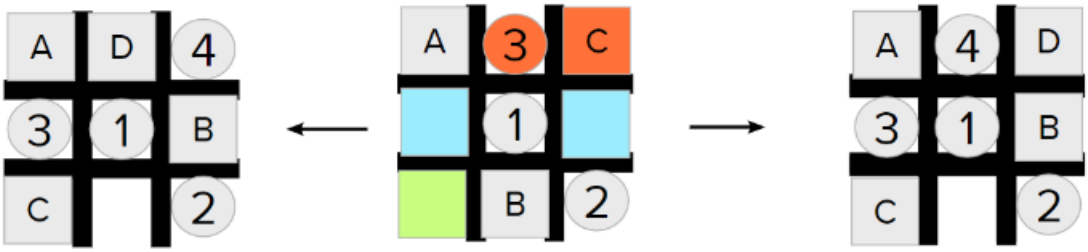

Figura 83: Cuarta posición esquina par E_C

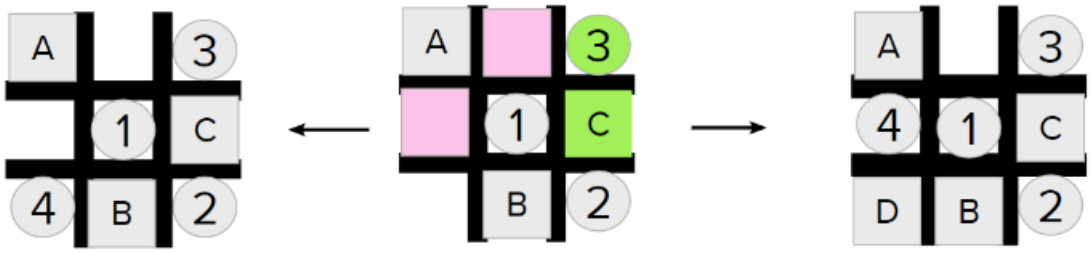

Figura 84: Quinta posición esquina par E_C

Los pasos por seguir para programar las combinaciones previamente mostradas son prácticamente los mismos a los mencionados en la programación E_A. Recordamos que los pasos a seguir son los siguientes:

1- Identificar la posición del tercer bloque puesto por el usuario. Para lograr lo mencionado podremos cambiar, dependiendo del caso en cuestión, el objeto de referencia de medición de distancias.
2- Una vez identificada la posición, comparar la esquina de inicio junto con la posición a los casos previamente presentados.
3- Tan pronto se identifique un caso, desplazar el tercer cilindro a la posición mostrada por la figura representante del caso.
4- Identificar la posición del cuarto bloque, en caso de que sea necesario variar el objeto de referencia de medición de distancias.
5- Una vez instaurada la condición, posicionar el cuarto cilindro de la manera que más nos convenga.

Los casos que quedan por describir precisan de una programación sencilla ya que, podemos observar que las combinaciones a tener en cuenta son bastante reducidas a comparación del caso anterior. En las siguientes ilustraciones (figura 85-90) mostraremos las combinaciones de dichos casos. Hemos de recalcar que es importante tener en cuenta que las esquinas pares e impares han de ser programas por separado ya que estas usan en diferente orden las variables de movimiento.

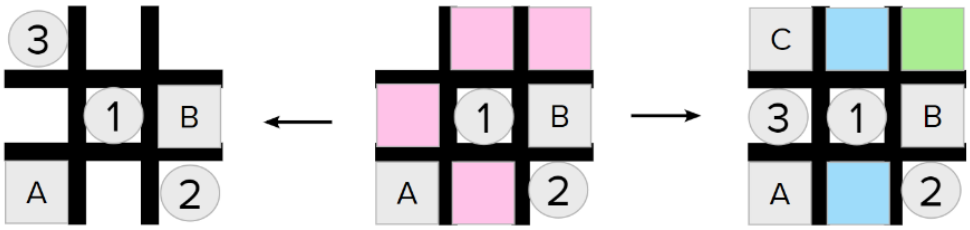

Figura 85: Combinaciones según la variable aleatoria 1 esquina impar

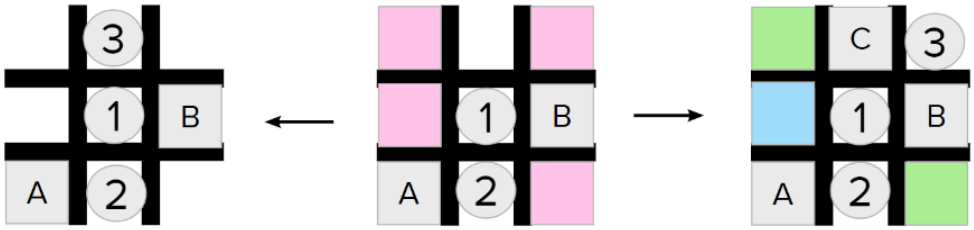

Figura 86: Combinaciones según la variable aleatoria 2 esquina impar

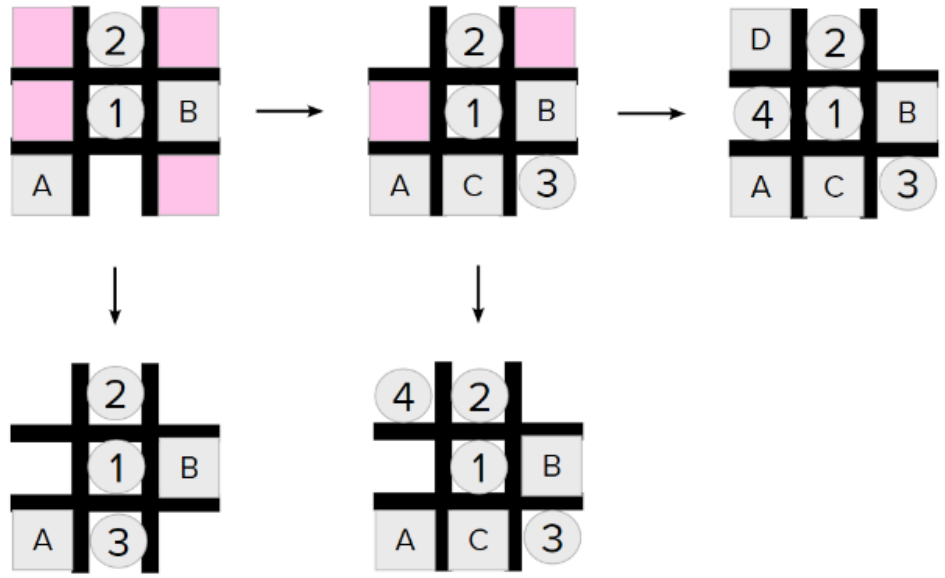

Figura 87: Combinaciones según la variable aleatoria 3 esquina impar

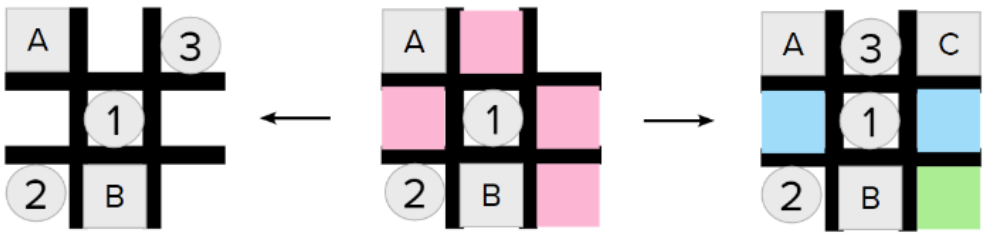

Figura 88: Combinaciones según la variable aleatoria 1 esquina par

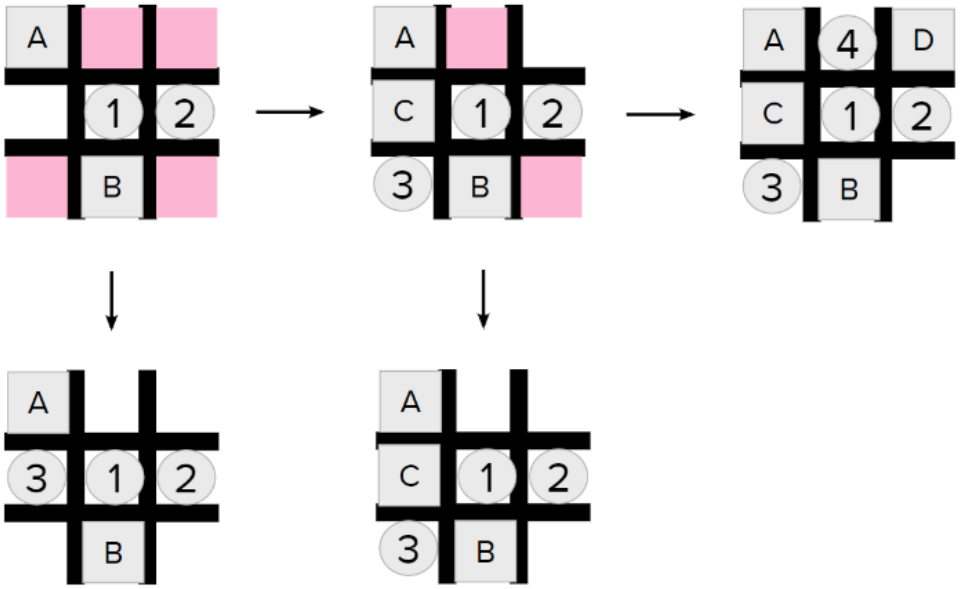

Figura 89: Combinaciones según la variable aleatoria 3 esquina par

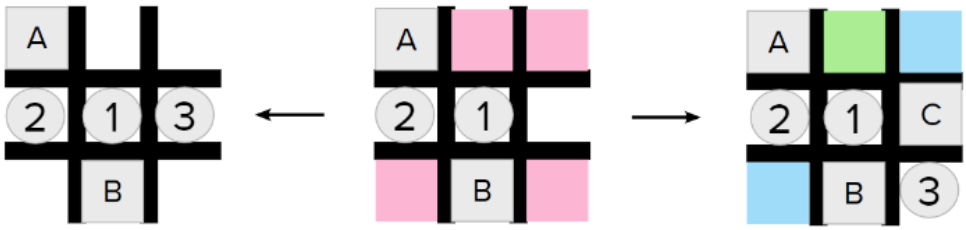

Figura 90: Combinaciones según la variable aleatoria 2 esquina par

Siguiendo los pasos a seguir explicados previamente, podemos desarrollar el código de las combinaciones posibles de la figura 89 como un ejemplo de programación. Una vez verificada que la posición del primer y segundo bloque satisfacen la posición ilustrada por la figura 89, posicionamos el segundo cilindro tomando como base Cam_x. Para identificar la posición del tercer cilindro podemos analizar si la distancia Dist_Y entre el bloque y la plantilla es inferior a 45mm. En caso de que esto último no se cumpla, podemos definir nuestra victoria desplazando el cilindro gracias a la variable de movimiento Camb_X. En caso contrario, desplazaremos el tercer cilindro a la posición definida por el uso de Camb_X y Camb_Y. Siguiendo lo explicado, para verificar la posición del cuarto bloque podemos hacer uso de una condición doble. Esta condición doble establecerá que, si la distancia Dist_Y con respecto al primer cilindro es inferior a 45mm y si la distancia Dist_X es superior a 95mm, podemos afirmar que el cuarto cilindro bloquea nuestra victoria. Si esta condición se cumple, tendremos que trasladar el cuarto cilindro usando Cam_Y. En caso de que dicha condición no se cumpla, marcamos nuestra victoria usando Cam_x y Cam_y.

3.2.4 Centro

En este apartado nos enfocaremos en describir la programación que se llevará a cabo una vez el primer bloque sea posicionado en el centro de la plantilla. Como está ilustrado en la figura 91, al posicionar el primer cilindro en cualquiera de los bordes podría desembocar una inevitable derrota. Es por esto por lo que, a la hora de desarrollar el estudio de las posibles combinaciones, el primer cilindro tendrá que siempre ser posicionado en alguna de las esquinas.

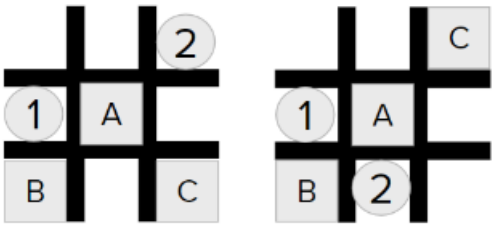

Figura 91: Ejemplos de combinaciones que llevan al fracaso

Es importante recalcar que, como misión de programación, buscamos llevar a cabo una programación que tenga en cuenta el mayor número de combinaciones posibles. Para lograr lo estipulado, como tomado en cuenta en las programaciones anteriores, tendremos en cuenta las diferentes esquinas en las que se posicionarán los cilindros. Las esquinas serán seleccionadas de manera aleatoria y las variables de movimiento serán actualizadas en base a la esquina de selección. Adicionalmente, se actualizarán los valores de las coordenadas de la coordenada 1. Generalmente

ésta última se refiriere a la posición del primer bloque puesto por el usuario, sin embargo, en este caso hará alusión a la posición de nuestro primer cilindro.

El código que se desarrollará bajo este apartado tendrá tres funciones separadas del código principal llamado "Centro" y dos funciones implementadas en el código principal. Para poder explicar de manera detallada las diferentes funciones que componen este apartado, separaremos las explicaciones por función como hecho en la explicación de "Esquina".

Centro_Rand

La función de programación que lleva por nombre Centro_Rand busca tomar en consideración las posibles combinaciones que toman lugar una vez el segundo bloque se posiciona en diagonal respecto al primer cilindro.

Con el objetivo de ilustrar las diferentes combinaciones posibles, se mostrarán ejemplos de ellas tomando en cuenta las diferentes esquinas y posiciones cruciales a tomar (figura 92). No obstante, algunas de estas opciones pueden llevar al fracaso, por lo que, si bien las ilustramos, no las desarrollaremos en la programación (figuras 93-94).

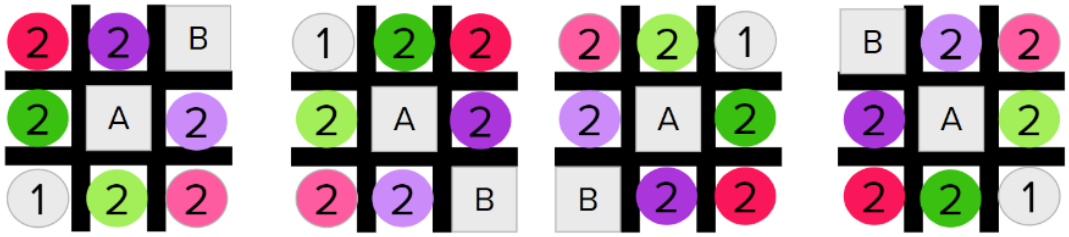

Figura 92: Ejemplos de combinaciones según las esquinas

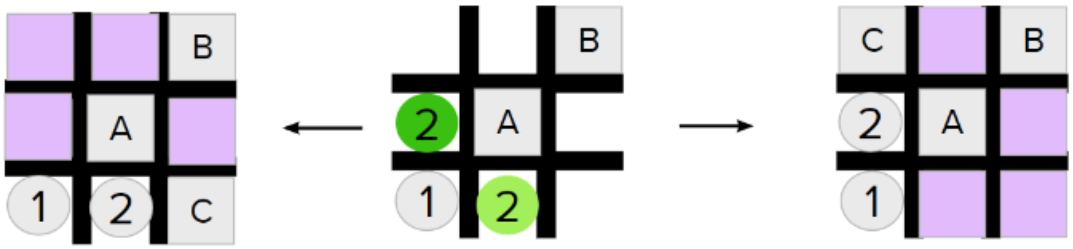

Figura 93: Combinaciones según tonos de verde

Nota: volvemos a denotar el significado de la pigmentación morada de las posiciones de la plantilla. Estos cuadrados, simbolizan las posiciones en las que podemos instaurar nuestro tercer cilindro sin impedir la victoria del oponente.

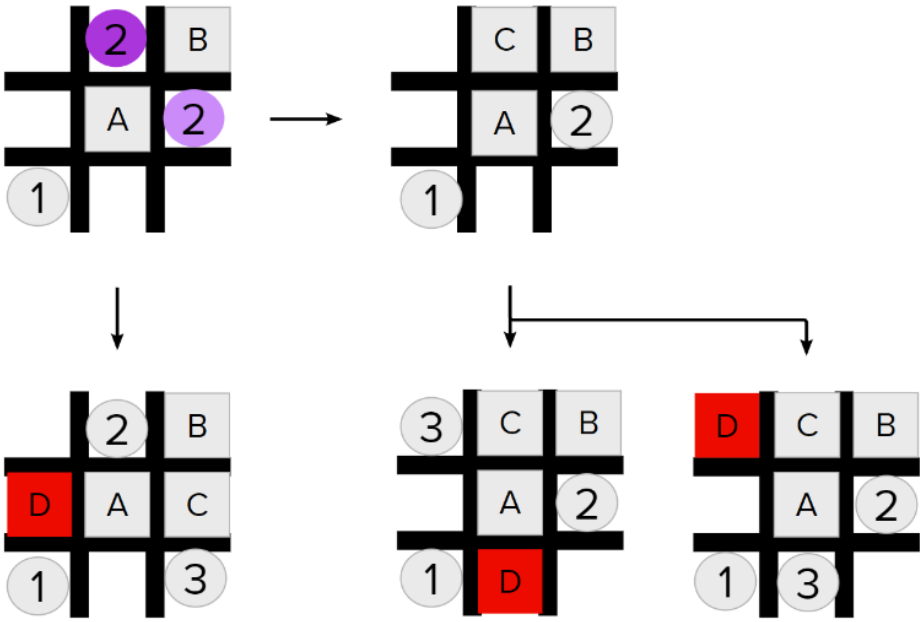

Figura 94: Combinaciones según tonos de morado

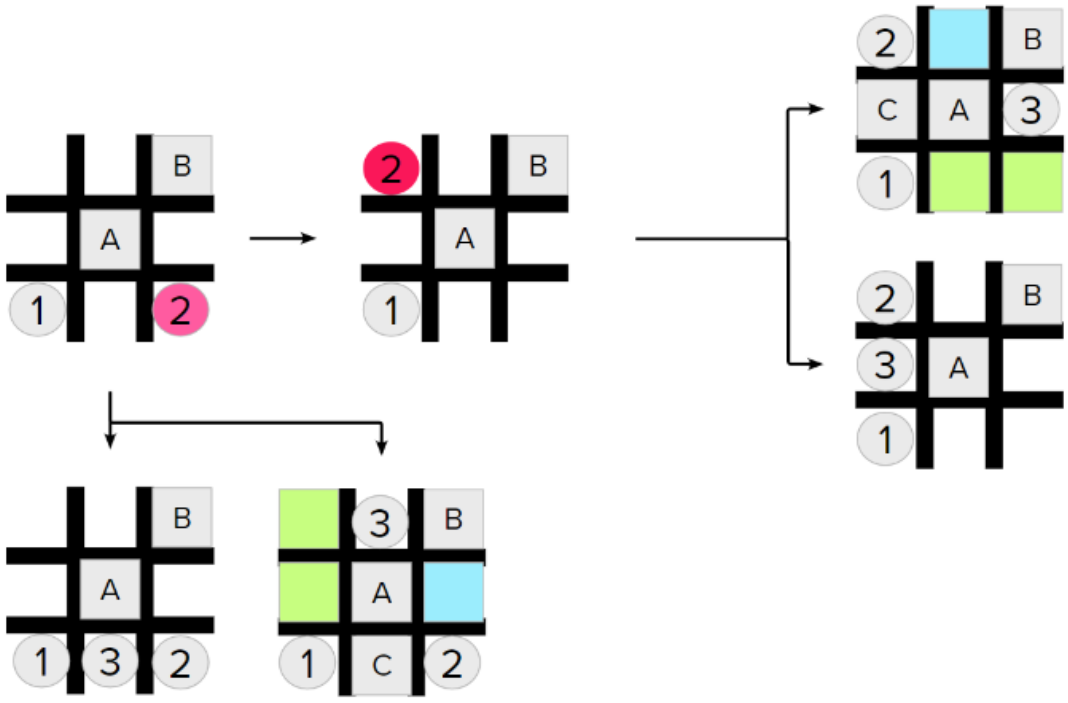

Figura 95: Combinaciones según tonos de fucsia

Como podemos observar en las figuras presentadas previamente, las únicas combinaciones que no brindan posibilidades de pérdida son las combinaciones ilustradas en la figura 95. Al tener sólo estas combinaciones posibles, el código reduce considerablemente en tamaño.

Tomaremos como ejemplo el primer caso descrito por la figura 95. Una vez hayamos verificado que la posición del primer y segundo bloque concuerdan con las posiciones descritas por la figura 94,

identificamos la posición del tercer bloque. Para ello, usaremos la variable Dist_Y tomando como referencia las coordenadas del primer cilindro. Si la variable Dist_Y tiene por valor uno inferior o igual a 45mm, quiere decir que dicho cilindro se encuentra bloqueando nuestra victoria. De no ser así, desplazaremos el tercer cilindro empleando Cam_y. En caso de que la condición se cumpla, trasladaremos el tercer cilindro usando la variables Camb_Y. Siguiendo este último caso, verificaremos la posición del cuarto bloque. Para ello usaremos la variable Dist_X tomando las coordenadas del primer cilindro como referencia. Si nos encontramos en el caso en el que Dist_X es inferior o igual a 65mm desplazamos el cuarto cilindro aplicando la variable Cam_x. En defecto de la previa condición, moveremos el cuarto cilindro usando Camb_X.

Centro_B_1_3 y Centro _B_2_4

Ambas funciones toman en consideración las combinaciones posibles que se pueden llevar a cabo una vez el segundo bloque se encuentre en alguno de los bordes. La principal diferencia que se puede encontrar entre estas dos funciones es la base de referencia de las esquinas. Concretamente, "Centro_B_1_3" tiene como base de programación las esquinas 1 y 3 (para referencia figura 45), mientras que, la función "Centro_B_2_4" usa como base las esquinas 2 y 4. Las combinaciones que se llevan a cabo una vez el usuario posiciona el segundo bloque en alguno de los bordes se pueden ver representadas en la figura 96. Adicionalmente, es importante notar que mostraremos las combinaciones que se dan lugar una vez colocamos el primer cilindro en la primera esquina. Esto se debe a que buscamos explicar lo crucial a la comprehensión del lector.

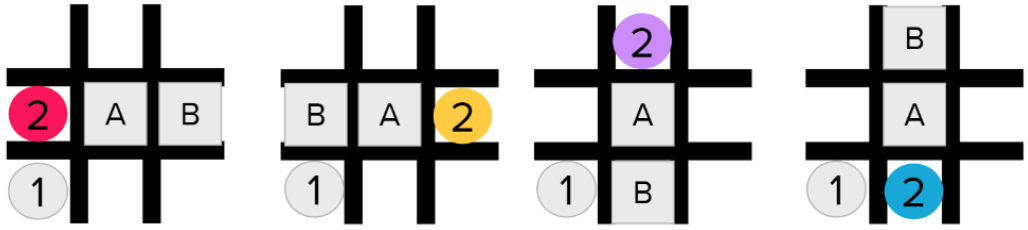

Figura 96: Combinaciones según Centro_B

Veremos que, no solo tendremos que tomar en cuenta las distancias Dist_X y Dist_Y en valor absoluto, sino que, tendremos también que crear una nueva condición que verifique la posición exacta del bloque. Para lograr lo último, verificamos los valores de las variables mencionadas en las posiciones mostradas en la figura 97.

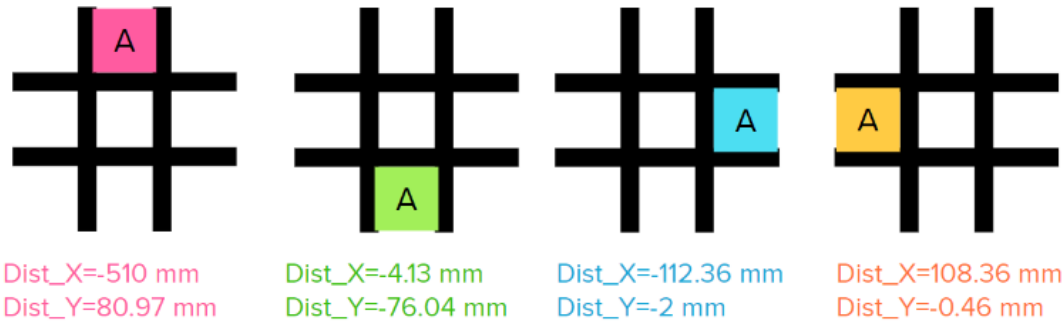

Figura 97: Valores de Dist_X y Dist_Y en posiciones específicas

Al comparar los valores previamente ilustrados con los valores recogidos de la puesta en práctica, podemos verificar fácilmente la posición de los bloques para saber exactamente en qué caso de la figura 96 nos encontramos.

Posteriormente, desarrollaremos las diferentes combinaciones que nacen del posicionamiento del segundo bloque en alguno de los bordes (figuras 98-101).

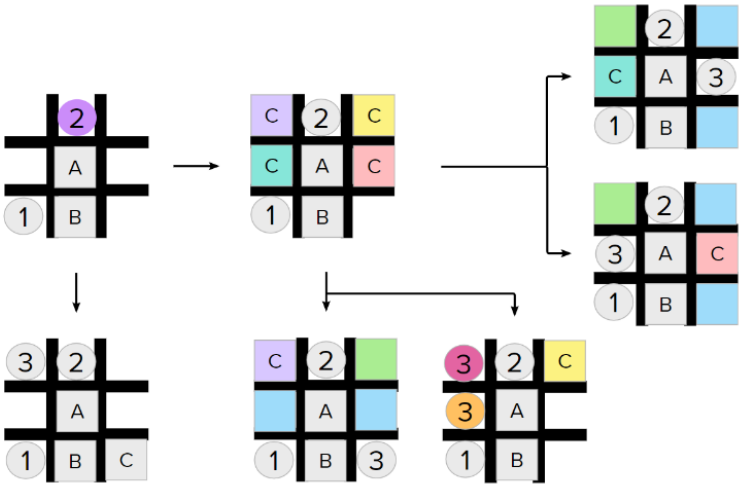

Figura 98: Combinación morada

Nota: las casillas celestes y verdes representan el caso en el que, si se introduce el bloque en una de ellas, se tendrá que introducir el cilindro en la casilla del otro color. Adicionalmente, podemos notar los cilindros de diferentes colores (fucsia y mostaza). Esto último implica la existencia de ambos casos en el código.

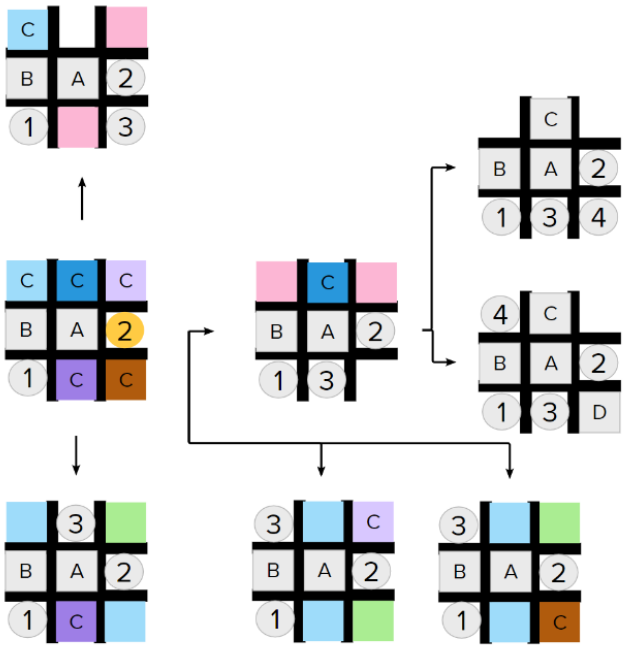

Figura 99: Combinación mostaza

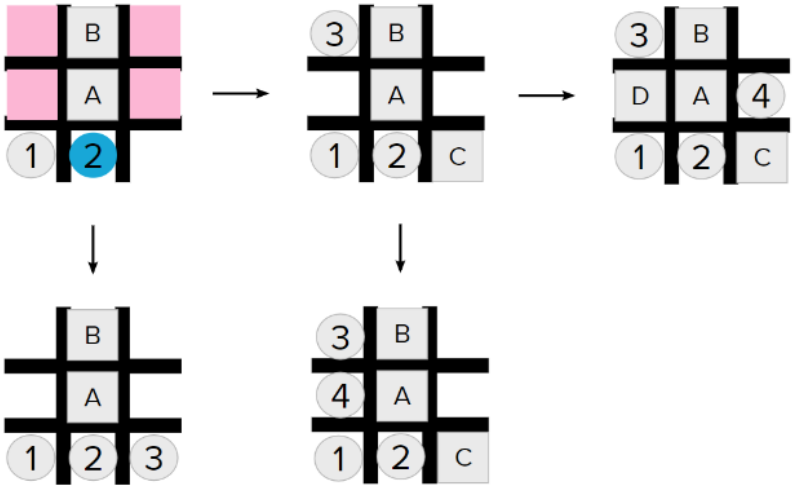

Figura 100: Combinación azul

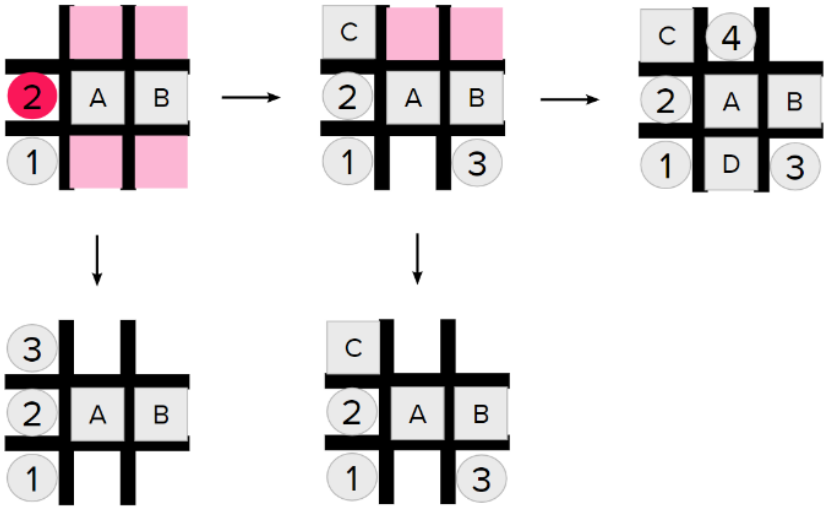

Figura 101: Combinación roja

Como se puede observar en las figuras previamente presentadas, mostramos solamente las combinaciones que se llevan a cabo una vez colocamos el primer cilindro en la primera esquina. Esto se debe a que, para conseguir las combinaciones de las esquinas pares es simplemente necesario rotar las representaciones gráficas a la derecha. Como se sabe, lo que principalmente separa las esquinas pares de las impares es el uso de las medidas de distancia Dist_X y Dist_Y.

Bajo el propósito de ilustrar tanto lo explicado previamente como la programación a desarrollar, explicaremos a detalle la programación de la combinación 101 para las esquinas 1 y 2.

Una vez hayamos confirmado que la posición del segundo bloque coincide con la figura 101, proseguimos a encontrar y desplazar el segundo cilindro bajo la variable de movimiento Camb_X. A fin de poder analizar correctamente la posición del tercer bloque, calcularemos las variables de distancia usando como referencia la posición en el eje X de nuestro primer cilindro. Más específicamente, si el valor de la distancia Dist_X es inferior o igual a 65mm podemos asumir que el bloque se encuentra en la misma vertical que el objeto de referencia. Si nos encontramos en el

caso en el que no se cumpla que la distancia Dist_X sea inferior o igual a 65mm, desplazamos el tercer cilindro gracias a Camb_X y Camb_Y para asegurar la victoria. En caso contrario, tendremos que desplazar nuestro tercer cilindro a la posición marcada por el uso de Cam_x y Cam_y. Siguiendo este último caso, lo que quedaría por verificar sería la posición del cuarto bloque. En caso de que el cuarto bloque se encuentre en la misma horizontal que nuestro primer cilindro indicaría un empate automático obligándonos a posicionar el cuarto cilindro bajo la variable Camb_Y. Si el cuarto bloque no se encontrase en la misma horizontal que nuestro primer cilindro, marcaríamos la victoria moviendo el cuarto cilindro gracias a Cam_y.

En caso de que nos encontremos con la posición del primer cilindro bajo la segunda esquina, es decir arriba a la izquierda (figura 45), comenzamos la programación verificando la posición del segundo bloque. Una vez hayamos verificado que nos encontramos en el caso de la figura 101 rotado a la derecha, desplazamos el segundo cilindro a la ayuda de la variable Cam_y. Para identificar la posición del tercer bloque, verificaremos el valor de la variable Dist_Y. En caso de que el tercer bloque se encuentre en la misma horizontal de nuestro primer cilindro, es decir bloqueando nuestra victoria, Dist_Y tendría que ser inferior o igual a 45mm. Al encontrarnos bajo la situación en que la condición anterior no se cumpla, desplazaremos en cilindro a la posición de victoria usando Cam_x y Cam_y. De no ser así, tendremos que desplazar el tercer cilindro a la posición denotada por las variables de movimiento Camb_X y Camb_Y. A fin de establecer nuestro siguiente movimiento tendremos que examinar la posición del cuarto cilindro. Al utilizar nuestro primer cilindro como objeto de referencia de medición de las variables de distancia podemos, de manera muy sencilla, verificar si el cuarto bloque se encuentra o no en la misma vertical. Suponiendo que el bloque se encuentre en la misma vertical, nuestra victoria se verá perjudicada por lo cual tendremos que hacer uso de la variable Cam_x para asegurar el empate. Suponiendo el caso opuesto, moveremos el cuarto cilindro gracias a la variable Camb_X cerciorando la victoria.

Funciones internas

Las combinaciones posibles que quedan por ser desarrolladas son las funciones que pueden ser encontradas dentro de la programación principal de "Centro". Dichas funciones nacen de los casos en los que el oponente coloque el segundo bloque en la esquina opuesta del primer cilindro tanto de manera vertical como horizontal (figura 102-103 respectivamente).

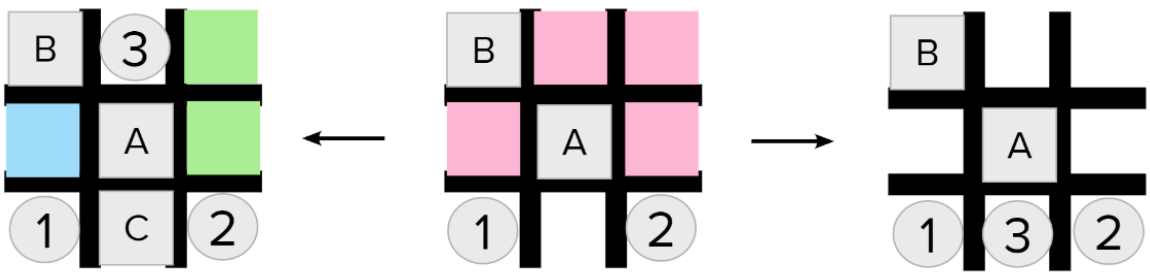

Figura 102: Combinación función vertical

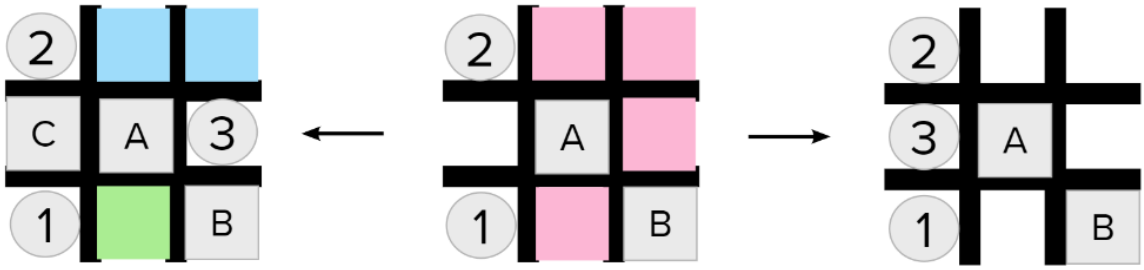

Figura 103: Combinación función horizontal

Como podemos observar, la programación necesaria a desarrollar para ambos casos es bastante sencilla, ya que, no presentan muchas combinaciones posibles. Para ilustrar la simplicidad de programación, explicaremos la programación a ser desarrollada para la figura 102.

Una vez se haya confirmado que las posiciones del primer y segundo bloque satisfacen las posiciones ilustradas en la figura 102, procedemos a trasladar el segundo cilindro a la posición definida por las variables Cam_x y Cam_y. Una vez posicionado el segundo cilindro, hacemos uso de la variable Dist_Y tomando como referencia las coordenadas del primer cilindro. De esta manera, se nos facilita el establecer la posición del bloque. En caso de que Dist_Y sea inferior o igual a 45mm, el bloque se encontrará bloqueando nuestra victoria obligándonos a trasladar el tercer cilindro empleando la variable Camb_Y. En caso contrario marcaremos la victoria desplazando el cilindro en cuestión utilizando Cam_y. Si nos encontramos en el caso anteriormente descrito, caso en el que nuestra victoria haya sido bloqueada, tendremos que identificar la posición del cuarto bloque. Para ello, tomaremos como referencia las coordenadas del primer cilindro. En caso de que la Dist_X sea inferior o igual a 65mm desplazaremos el cuarto cilindro a la ayuda de la variable Cam_x. De no ser así, es decir que la condición previamente establecida no se cumpla, hemos de trasladar el cilindro utilizando Camb_X.

3.2.5 Borde

Una vez analizados los casos en los que el usuario comenzase tanto en la esquina como en el centro, sólo nos queda por analizar el caso en el que comience por los bordes. Una nos encontremos en mencionado caso, el robot tendrá dos opciones: colocar el cilindro en alguna esquina o en el centro. En caso de que coloquemos el cilindro en el centro, logramos bloquear dos líneas que lleven a una victoria inminente del usuario. Es por esto último que la estrategia a ser puesta en práctica es la de posicionar el cilindro en el centro de la plantilla.

La programación de este apartado se realizará de manera particular puesto que, cambiaremos constantemente las coordenadas de referencia para las mediciones de las variables Dist_X y Dist_Y. De la misma manera, se cambiarán las esquinas de referencia dependiendo del caso en el que nos encontremos para intentar conseguir una programación eficiente. A fin de ilustrar más profundamente lo estipulado, mostraremos tanto las combinaciones posibles de manera general como ciertos casos específicos.

B_N_C

Esta función nos permite analizar las combinaciones posibles que se pueden dar una vez se ponga el primer bloque en la horizontal central de la plantilla. Bajo el propósito de lograr un programación eficiente y sencilla, dentro de lo que cabe, buscamos realizar la numeración de los casos y la

selección de las respectivas esquinas de la manera más simétrica posible. En la figura 102 podremos observar la numeración que tomaremos a la hora de desarrollar los diferentes casos.

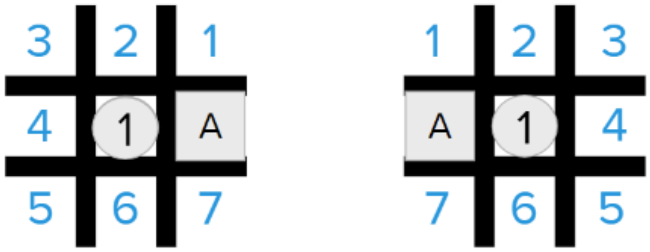

Figura 104: Numeración de casos en la horizontal

Como primer objeto de análisis, tomaremos las diferentes combinaciones posibles en los casos 3 y 5 mostrados en la figura 104. Al observar las figuras a continuación (figura 105 y 106), podemos fácilmente ver la simetría que se presenta entre los diferentes casos.

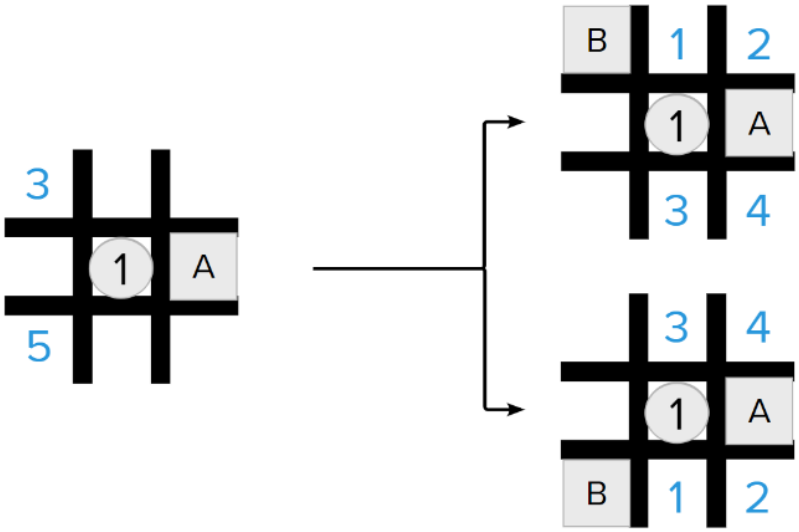

Figura 105: Casos derivados de 3 y 5 en configuración derecha

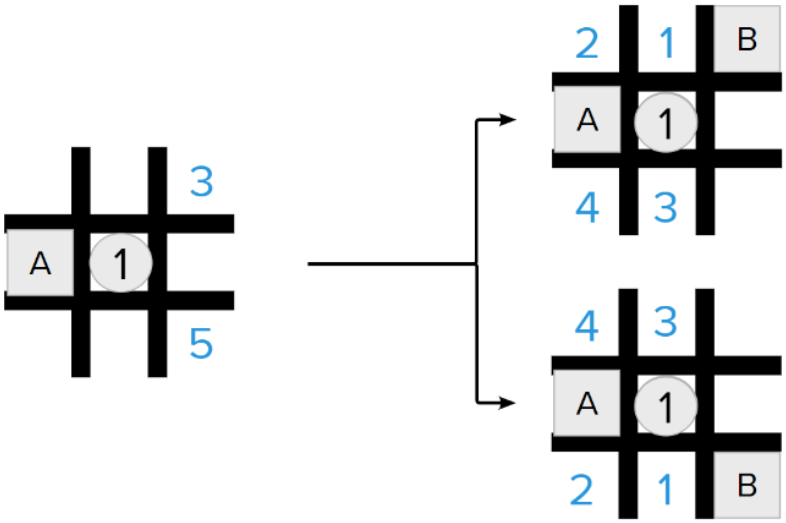

Figura 106: Casos derivados de 3 y 5 en configuración izquierda

Bajo el propósito de volver todo lo más eficiente y sencillo posible, para estos casos tomaremos las siguientes consideraciones:

- Al estar en la situación 3 de la configuración derecha, tomaremos la esquina número dos como la esquina de referencia para las variables de movimiento (ejemplo figura 107).
- Al estar en la situación 5 de la configuración derecha, tomaremos la primera esquina como referencia para las variables de movimiento (ejemplo figura 109).
- Al estar en la situación 3 de la configuración izquierda, usaremos la tercera esquina como referencia para las variables de movimiento (ejemplo figura 108).
- Al estar en la situación 5 de la configuración izquierda, usaremos la cuarta esquina como referencia para las variables de movimiento (ejemplo figura 110).

Para ilustrar el razonamiento y la ejecución de las condiciones previamente explicadas, mostraremos un ejemplo de las combinaciones y de la programación a ser desarrollada.

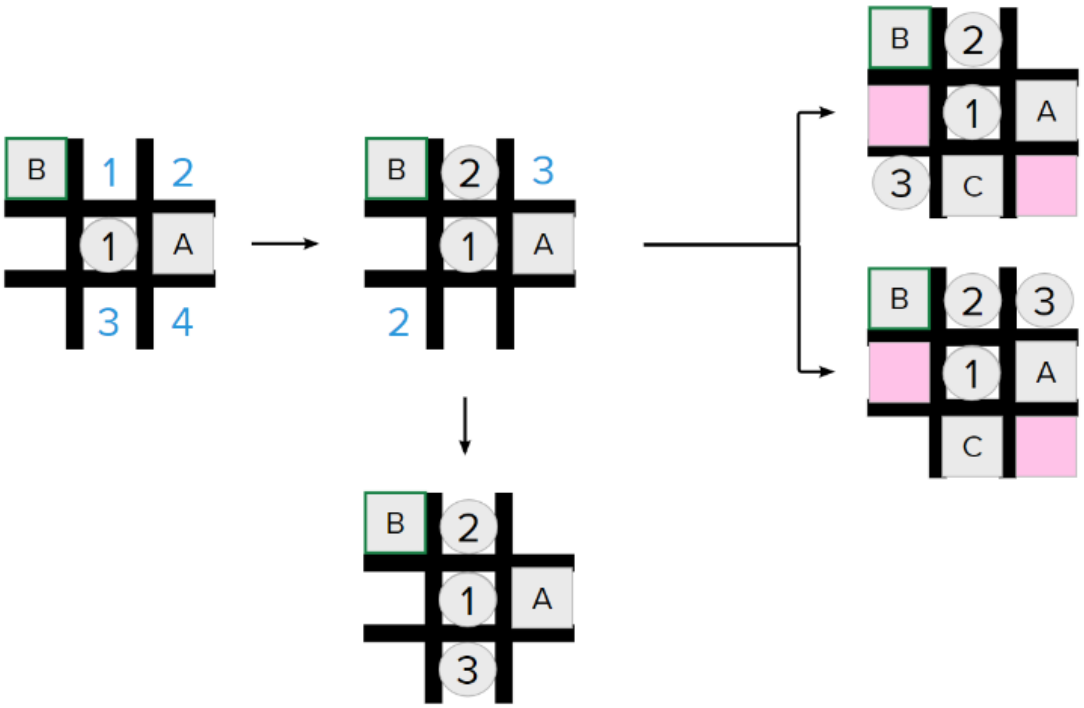

Figura 107: Ejemplo caso 3 configuración derecha

Nota: es importante recalcar que el encuadre color verde oscuro indica la esquina de referencia para las variables de movimiento.

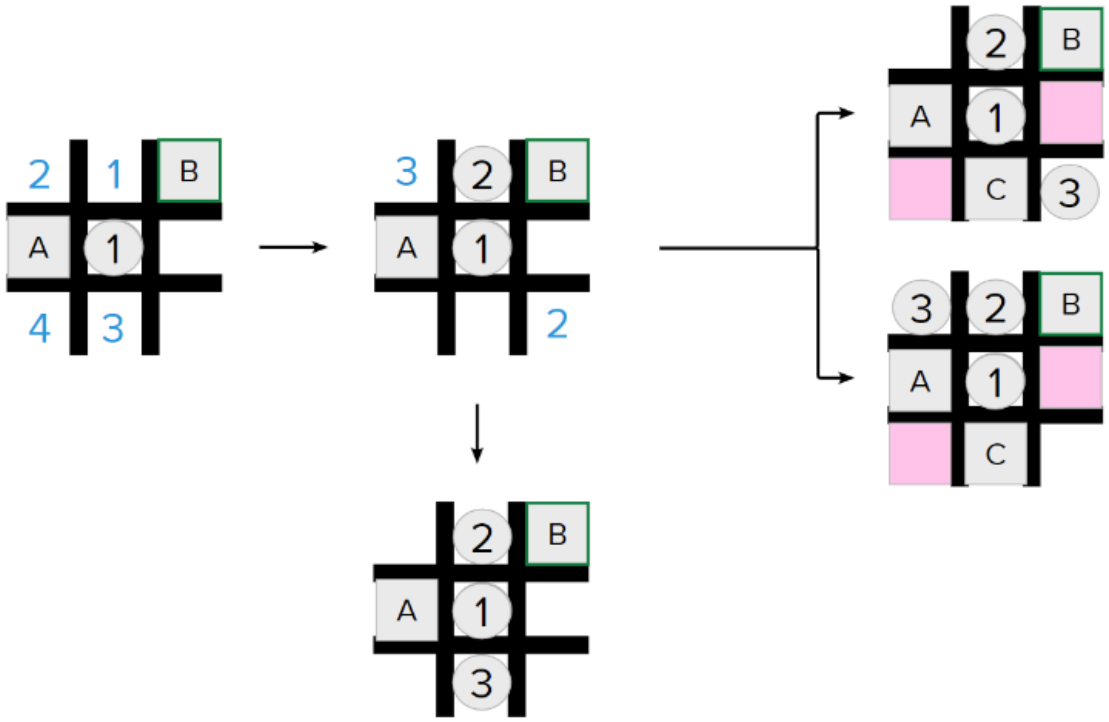

Figura 108: Ejemplo caso 3 configuración izquierda

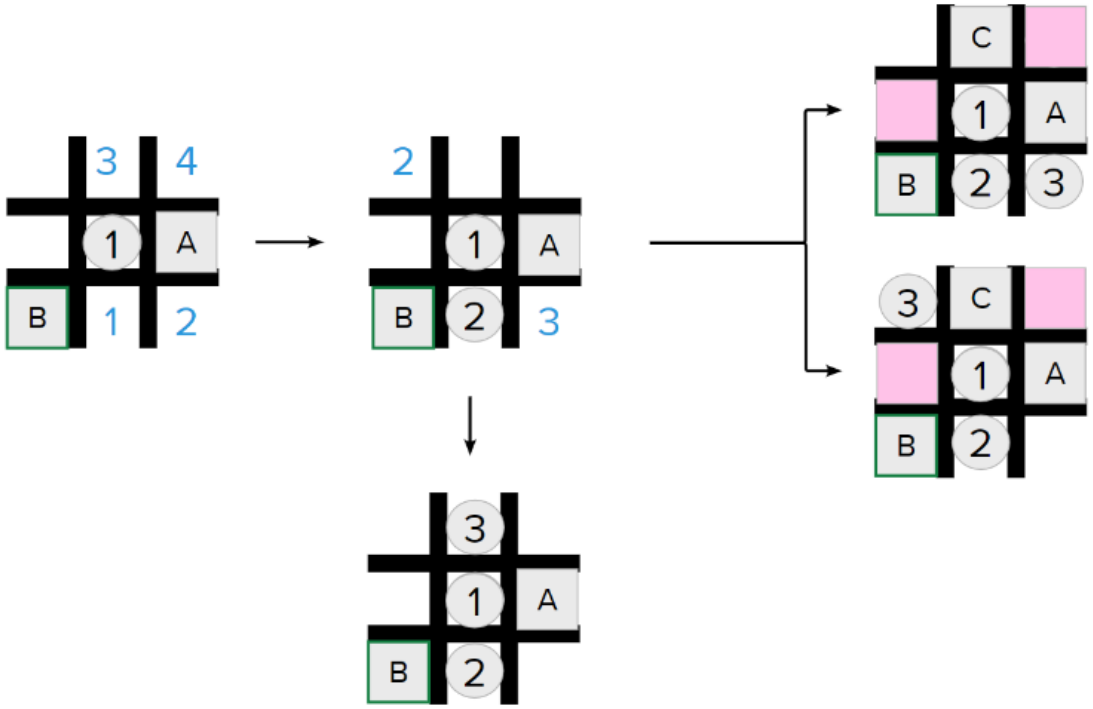

Figura 109: Ejemplo caso 5 configuración derecha

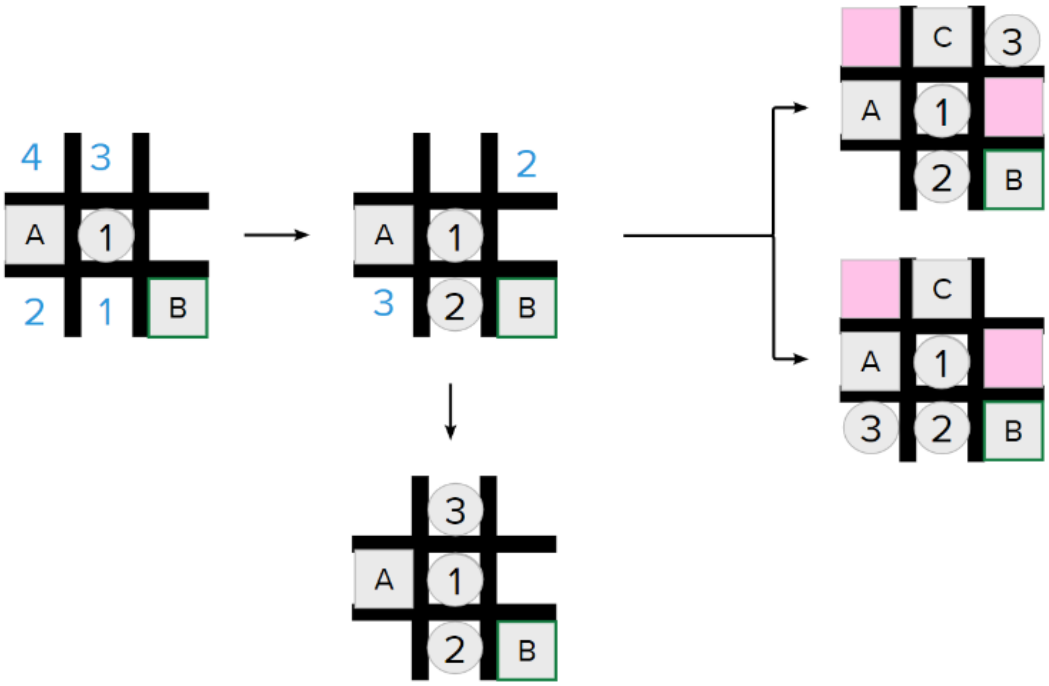

Figura 110: Ejemplo caso 5 configuración izquierda

Como se ha mencionado en apartados anteriores, los pasos a seguir para programar las diferentes combinaciones son los mismos. Una vez seguidos estos pasos, podemos hacer una tabla marcando las variables de movimiento a ser utilizadas (tabla 1). Es fácil comprobar que, al haber escogido las diferentes esquinas de referencia, se cumple que el orden de las variables de movimientos a ser utilizadas es el mismo.

	Camb_X	Camb_Y	Cam_x	Cam_y
Segundo cilindro	-	-	-	X
		Ganamos		
Tercer cilindro	-	X	-	-
		Bloqueado (3)		
Tercer cilindro	X	X	-	-
		Ganamos (3)		
Cuarto cilindro	-	-	X	X
		Bloqueado (3)		
Cuarto cilindro	-	X	X	-

Tabla 1: Variables de movimiento caso 3

El razonamiento será el mismo a lo largo de la programación. Más específicamente, se cambiarán las esquinas de referencia con tal de asegurar un directo paralelismo a la hora del uso de las variables de movimiento.

A continuación, se mostrarán el resto de las combinaciones posibles recalcando en cada una de ellas las esquinas que se tomarán como referencia para las variables de movimiento (figuras 111-114).

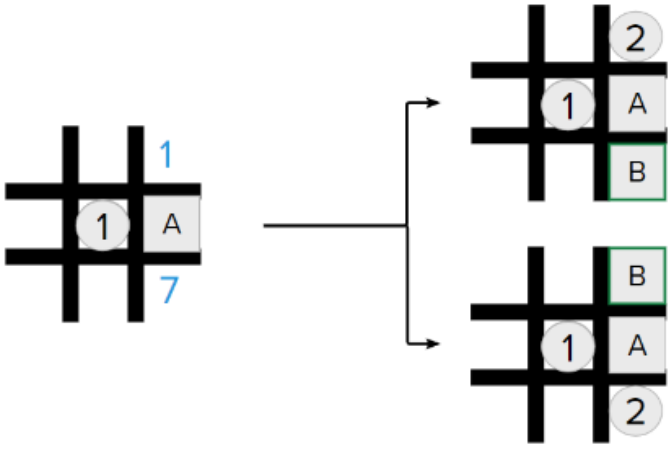

Figura 111: Casos derivados de 1 y 7 en configuración derecha

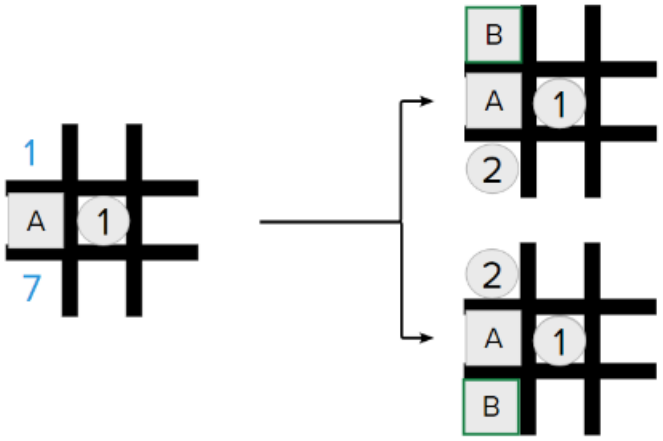

Figura 112: Casos derivados de 1 y 7 en configuración izquierda

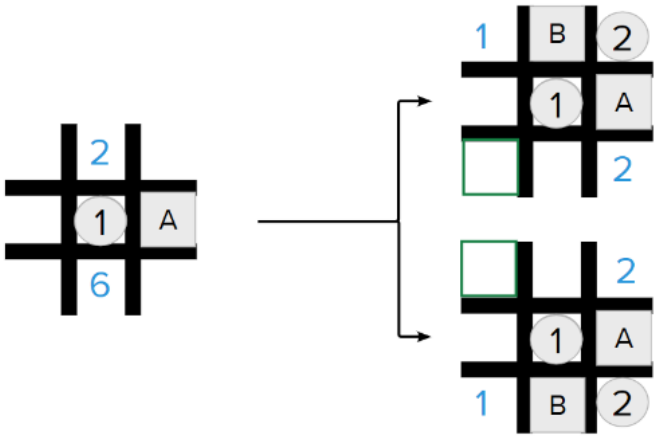

Figura 113: Casos derivados de 2 y 6 en configuración derecha

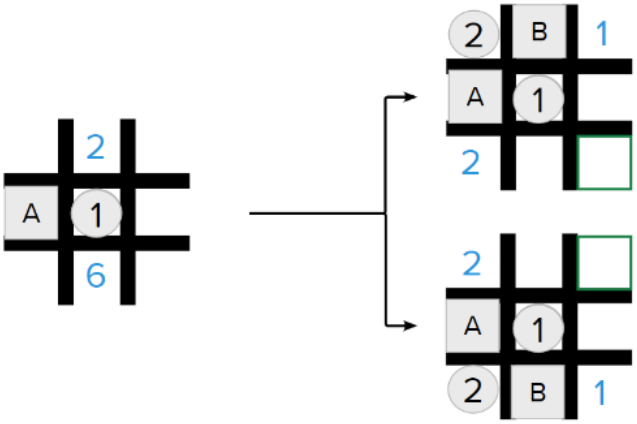

Figura 114: Casos derivados de 2 y 6 en configuración izquierda

Si bien explicamos anteriormente que el resto de la programación se realizaría de misma forma, cabe a resaltar un caso en particular, el cuarto caso. Como podemos ver en la figura 115, nos encontramos en un caso que presenta un alto parecido entre la configuración izquierda y derecha. Dado que ambas configuraciones son especialmente parecidas, utilizaremos la misma programación para ambas. No obstante, hemos de notar la importancia de la diferencia en las posiciones del primer y segundo bloque.

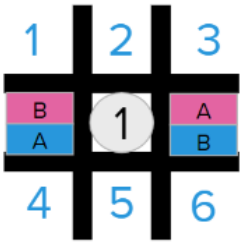

Figura 115: Caso 4 en ambas configuraciones

Para poder ilustrar las diferentes consideraciones para tener en cuenta por caso, construimos siguiente tabla.

CASOS	ESQUINA DE REFERENCIA
1	4
2	3
3	3
4	3
5	1
6	4

Tabla 2: Subcasos y esquinas de referencia del caso 4

Tomado un caso más específico, podemos explicar el sexto caso para ambas situaciones (figura 116). En dicha figura podemos observar como lo único que cambia es la posición del primer y segundo bloque. Esto afectará solamente a la hora de definir las condiciones de los distintos casos.

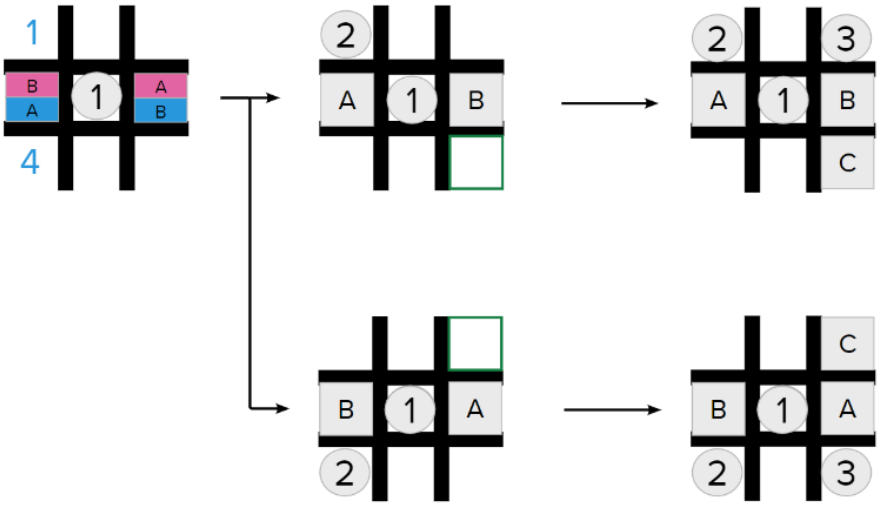

Figura 116: Subcasos 1 y 4 del cuarto caso en ambas configuraciones

Como hecho en casos anteriores, desarrollaremos una tabla que establezca las variables de movimiento a ser utilizadas. Es fácil ver que las variables de movimiento utilizadas son generales a ambos casos, lo único que difiere es el establecimiento de las condiciones.

	Camb_X	Camb_Y	Cam_x	Cam_y
Segundo cilindro	-	X	X	-
		Ganamos		
Tercer cilindro	X	-	-	X
		Bloquea		
Tercer cilindro	X	X	-	-

Tabla 3: Variables de movimiento subcasos 1 y 4 del cuarto caso

B_F_V

La mencionada función nos ayuda a trabajar con las combinaciones que se dan una vez se coloque el primer bloque en la vertical central de la plantilla. Como hecho en el apartado anterior, buscaremos tanto numerar los casos posibles de manera simétrica como seleccionar las esquinas de referencia de manera eficiente. En la figura 117 podremos observar la numeración que tomaremos a la hora de desarrollar los diferentes casos.

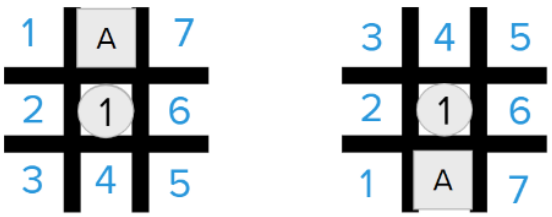

Figura 117: Numeración de los casos en la vertical

El pensamiento y desarrollo del código de este apartado se llevará de la misma manera que el del apartado anterior. Es por esto por lo que presentaremos una tabla con los casos posibles y las esquinas a ser consideradas como referencia para las variables de movimiento (tabla 4).

	Esquina de referencia	
Casos	Arriba	Abajo
1	4	3
2	4	3
3	1	2
4	-	-
5	4	3
6	1	2
7	1	2

Tabla 4: Casos y esquinas de referencia B_F_V

Para ilustrar lo expresado en la tabla 4 mostraremos las combinaciones que se dan una vez nos encontramos en los casos 1 y 7 en las configuraciones arriba y abajo (figuras 118 y 119 respectivamente). De la misma manera en la tabla 5 veremos las variables de movimiento a ser utilizadas de manera generalizada para los casos mencionados.

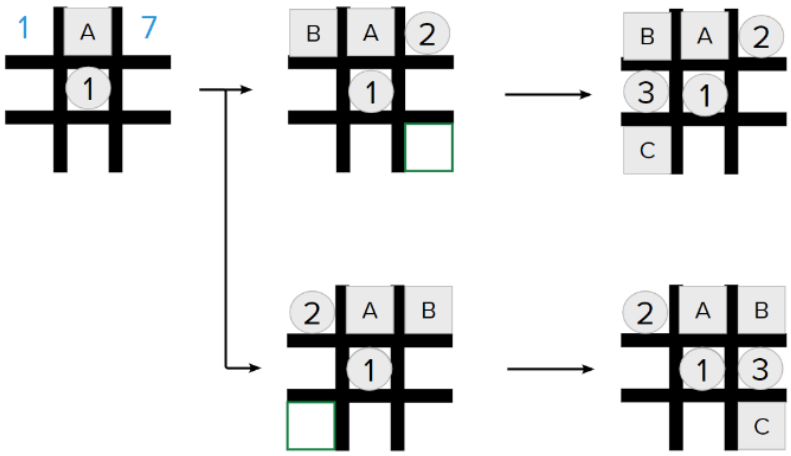

Figura 118: Casos 1 y 7 de la configuración arriba

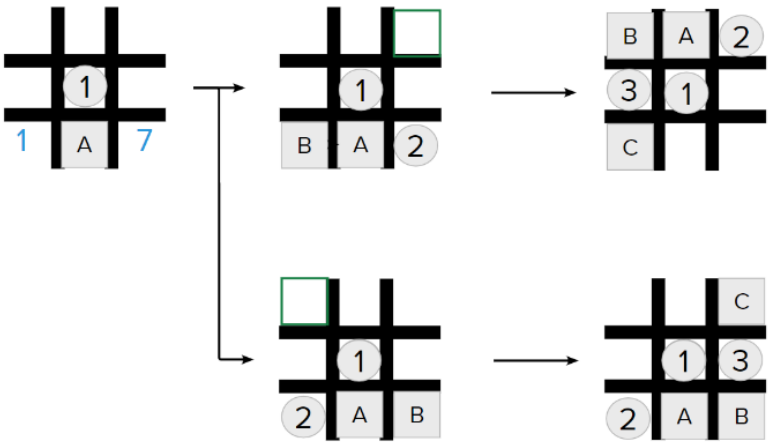

Figura 119: Casos 1 y 7 de la configuración abajo

	Camb_X	Camb_Y	Cam_x	Cam_y
Segundo cilindro	X	X	-	-
		Ganamos		
Tercer cilindro	-	-	X	X
		Bloquea		
Tercer cilindro	-	-	X	-
		Ganamos		
Cuarto cilindro	X	-	-	-

Tabla 5: Variables de movimiento casos 1 y 7 de la función B_F_V

Se puede verificar fácilmente que las variables expuestas en la tabla 5 funcionan perfectamente en los casos ilustrados en las figuras 118 y 119.

El cuarto caso de la función B_F_V, como el cuarto caso de la función B_N_C, es un caso que representa un alto parecido entre ambas configuraciones. Es por ello por lo que lo describiremos de manera separada al resto. Si bien usaremos la misma programación para las configuraciones arriba y abajo, deberemos tener presente de manera constante el hecho de que las posiciones de los bloques en la vertical central de la plantilla difieren entre sí. La numeración que se realizará para describir los diferentes subcasos del caso 4 es la ilustrada en la figura 120. De misma manera, las esquinas de referencia que se tendrán en los diferentes subcasos se ven descritas en la tabla 6.

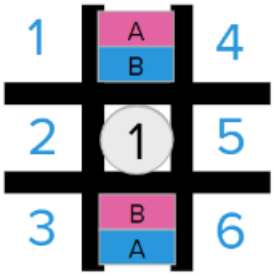

Figura 120: Subcasos del cuarto caso de la función B_F_V

Casos	Esquina de referencia
1	4
2	1
3	3
4	1
5	4
6	2

Tabla 6: Subcasos y esquina de referencia del cuarto caso de la función B_F_V

3.4 Garra

En este breve apartado buscaremos explicar las funciones básicas para la programación de la garra. Para ello, mostraremos las funciones llamadas "Pos" del código. Gracias a estas funciones, lograremos desplazar el robot hasta el segundo puesto de trabajo en el cual recogerá los cilindros para luego colocarlos en la posición inicial. A fin de explicar lo propuesto, mostramos a continuación el código desarrollado (figuras 121 y 122).

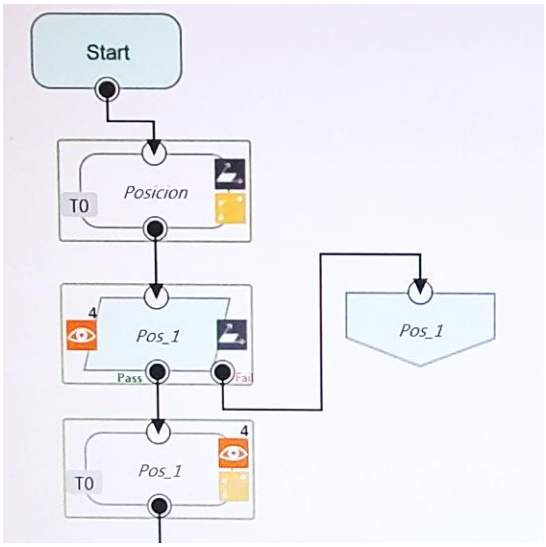

Figura 121: Primera parte código de la función "Pos_1"

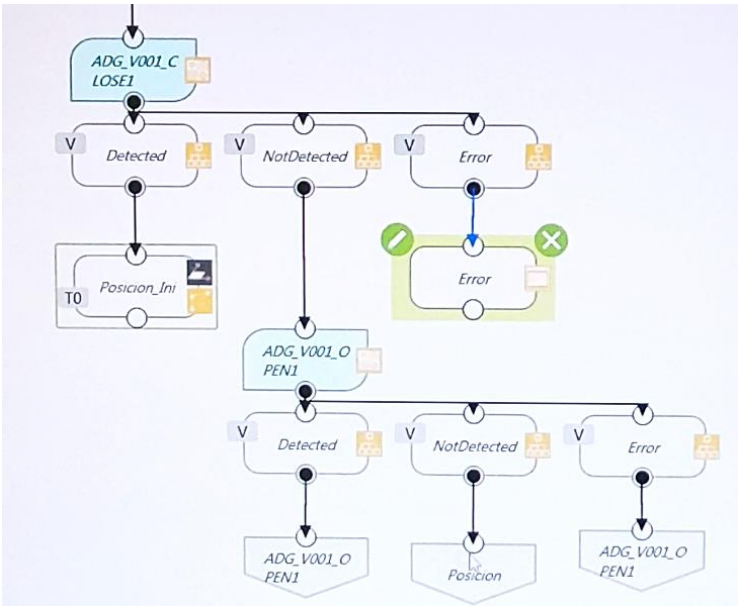

Figura 122: Segunda parte código de la función "Pos_1"

En la figura 121 podemos observar la parte del código que hace que el robot se desplace al segundo puesto de trabajo y busque el cilindro en cuestión, en este caso el primero. Analizando la figura 122 podemos ver como la programación de la garra no se reduce solamente a cerrar la garra. Esto se debe a que, la garra con la que trabajamos contiene sensores para determinar si la garra ha agarrado algo o no, por lo tanto, tenemos una condicionante que necesitará de pasos adicionales. Para programar el caso en el que la garra se cierre sin contener el cilindro entre las hazas, colocaremos el bloque que nos permita abrir nuevamente la garra y una vez esté abierta, volveremos al bloque inicial de la programación (figura 121).

Programando la garra de la manera expuesta nos aseguramos de cubrir los errores posibles, logrando de este modo, fijar un funcionamiento con la capacidad de autocorrección en caso de fallo.

3.5 Funcionamiento general

Una vez se haya llevado a cabo la programación de las diferentes funciones que conforman el programa, podemos explicar el funcionamiento general del sistema. Como bien sabemos el robot estará siempre esperando a que el usuario posicione el bloque en cuestión sobre la plantilla. Una vez el robot haya identificado la posición del bloque, ejecutará la función específica al caso en el que se encuentre. A fin de posicionar el cilindro que se requiera, el robot se desplazará al segundo puesto de trabajo, recogerá el cilindro y lo desplazará a la posición impuesta por las variables de movimiento utilizadas. Estos pasos se verán repetidos hasta que lleguemos, ya sea a una victoria o a un inevitable empate. Una vez terminado el juego, el cilindro guardará cuidadosamente los bloques y cilindros en el correcto orden. En la figura 123 podremos ver ilustrado lo previamente expuesto.

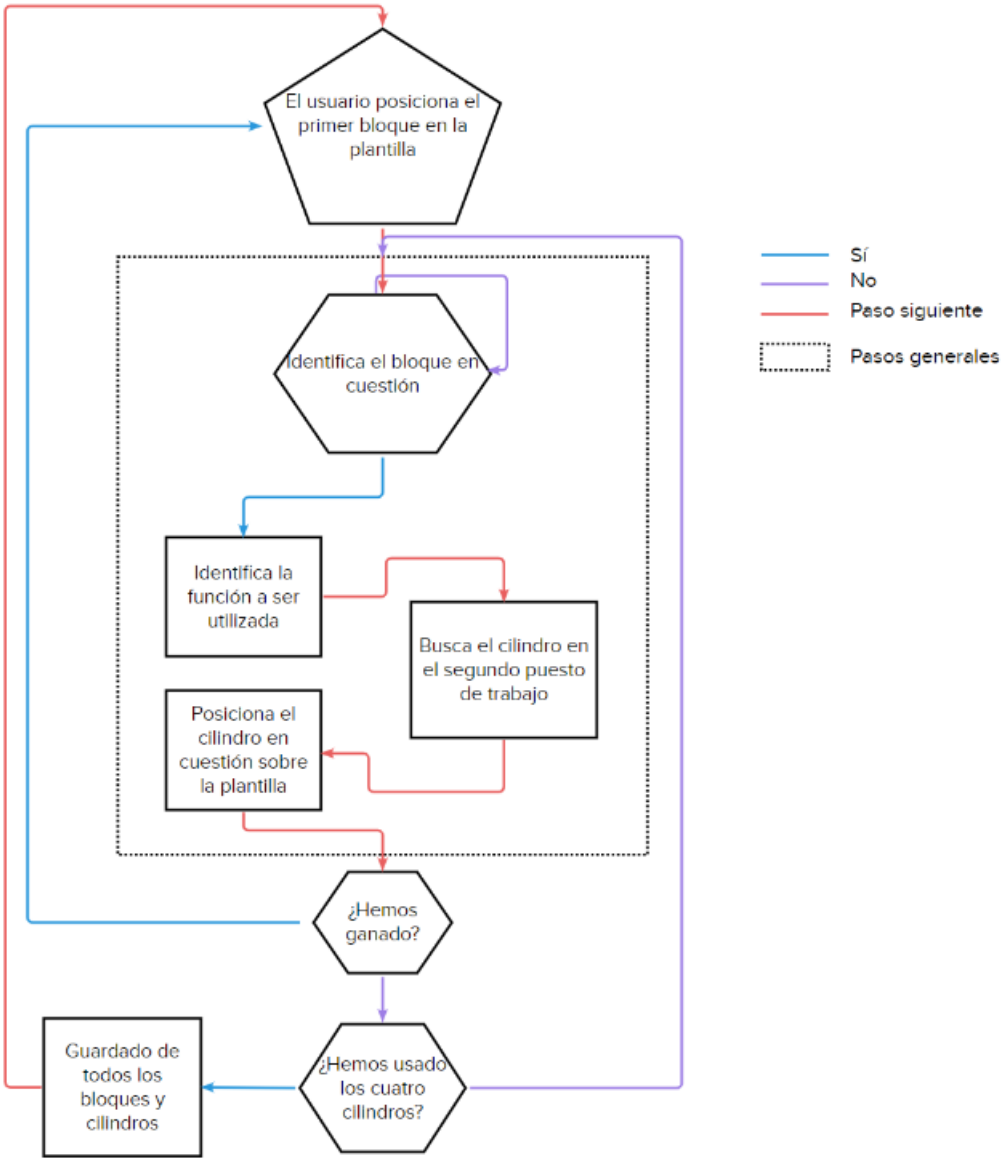

Figura 123: Diagrama de flujo del sistema

3.6 Errores

Dado que la programación del robot se realiza en modo bloque, la visualización de los errores cometidos se vuelve más complicada. Adicionalmente, la cámara con la que se trabaja no contiene una resolución perfecta, por lo que, las visiones que se vayan a realizar con dicha cámara no serán exactas.

Apoyando lo primeramente expuesto, recalcamos un error al momento de la definición de las bases. Como se ha explicado en los apartados anteriores, las definiciones de las bases son muy importantes a la hora de utilizar los nodos de visión, ya que, cada nodo de visión tendrá su propia base definida. Al tener cada nodo de visión su propia base, es posible que la base de referencia del robot cambie de la preestablecida a la base definida por el dicho nodo de visión. Al no darnos cuenta de esto y al desarrollar otros nodos de visión bajo otra base que no sea la del robot podremos ver como el robot va a posiciones de calibración no establecidas que variarán conforme se ejecute el código.

Por segunda parte, la cámara que utilizamos con la calibración que utilizamos debería asegurarnos un muy buen funcionamiento. No obstante, podemos notar que incluso después de realizar la configuración de los nodos de visión de la manera más cuidadosa, se pueden de todas formas, apreciar errores en la aproximación del robot al objeto en cuestión.

Si bien el primer error expuesto reside más en la comprensión y la familiarización del personal con el robot, este puede presentar varios retrasos. Esto último se puede explicar ya que los manuales de apoyo no presentan una solución o una referencia directa al sujeto.

3.7 Conclusiones del programa

El programa expuesto no solo nos permitió aprender a familiarizarnos con una interfaz de programación diferente a las conocidas previamente, sino que, nos permitió, a la vez, desarrollar una visión enfocada a la programación.

Bajo el mismo punto de vista, podemos ver como en el último apartado de la programación se tuvo que innovar constantemente la manera en la que nos dirigimos al problema con tal de conseguir la solución más eficiente.

Esta primera aproximación al mundo de la robótica nos ayuda a familiarizarnos con los problemas que se encuentran en esta área y desarrollar nuestro pensamiento a uno resolutivo y autodidáctico.

4.RETRATO

4.1 Acercamiento al rostro humano

A fin de poder dibujar de manera relativamente aproximada el rostro del usuario, tendremos primeramente que definir las bases sobre las cuales realizaremos los trazos. Para lograr lo expuesto, usaremos el canon del rostro humano. Canon, bajo este contexto, refiere al conjunto de normas que ayudan a dibujar, esculpir y pintar formas naturales, guardando en todo momento una cierta proporción entre las diferentes partes que las conforman. Cabe a recalcar que las proporciones en ciertos casos pueden variar, no obstante, nos limitaremos a usar los acercamientos más generales.

Como bien sabemos, la geometría puede ser encontrada en todos lugares en la naturaleza, y el rostro humano es un claro ejemplo de ello. Antes mencionamos las normas que se pueden usar para conservar la proporción en lo dibujado, esto se puede ver claramente ilustrado por la figura 124 que muestra el rostro humano perfectamente simétrico.

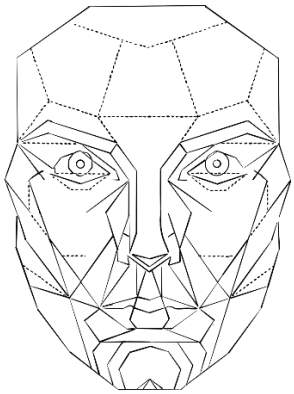

Figura 124: Rostro geométricamente perfecto

Si bien mostramos una imagen de la geometría perfecta en el rostro humano, sabemos de antemano que esto no se presenta siempre en la naturaleza. Es más, se lograron definir alrededor de 7 tipos de rostros en la naturaleza del ser humano (figura 125).

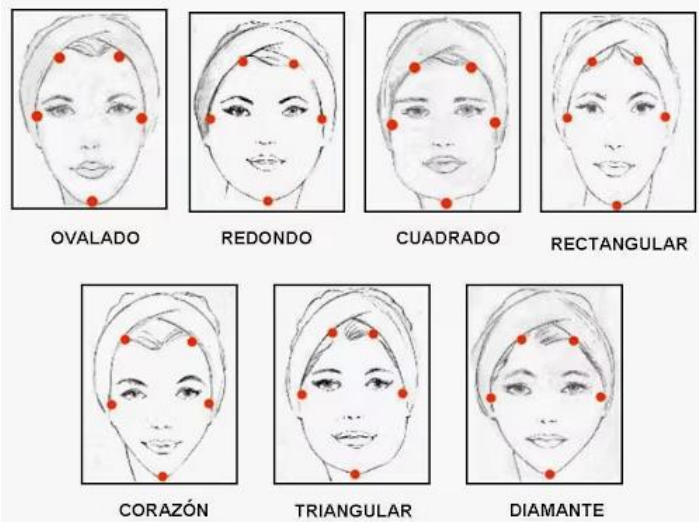

Figura 125: Tipos de rostros

La programación del código se enfocará en dibujar de manera aproximada los rostros humanos basándose tanto en los siete tipos de rostros diferentes como las normas previamente mencionadas. Para lograr lo anteriormente presentado, usaremos distintas figuras en posiciones clave del rostro para guardar sus coordenadas y definir tanto el tipo del rostro del que se trata como las longitudes y distancias de los trazos. Esto se podrá ver ilustrado en las figuras 126 y 127.

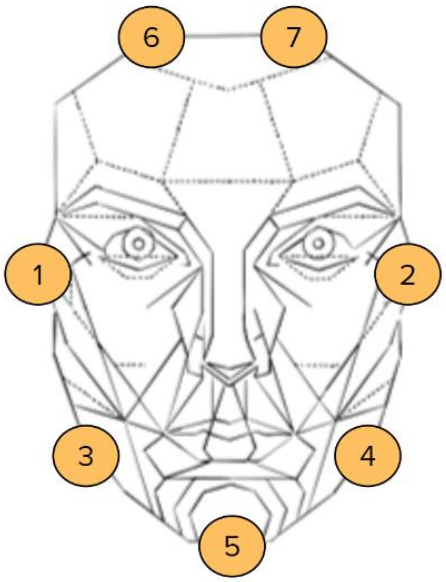

Figura 126: Posicionamiento de las figuras para el contorno del rostro

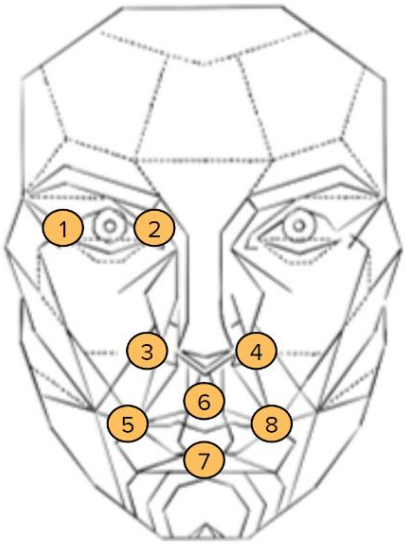

Figura 127: Posicionamiento de las figuras para el interior del rostro

4.2 Calibración y programación de la cámara

Como se ha mostrado en el apartado 3.2.1, la programación y calibración de la cámara se realizará de la misma manera. No obstante, en este caso tendremos que implementar reconocimiento de diferentes formas a las utilizadas previamente. Las formas que se buscará reconocer son las presentadas en la figura 128.

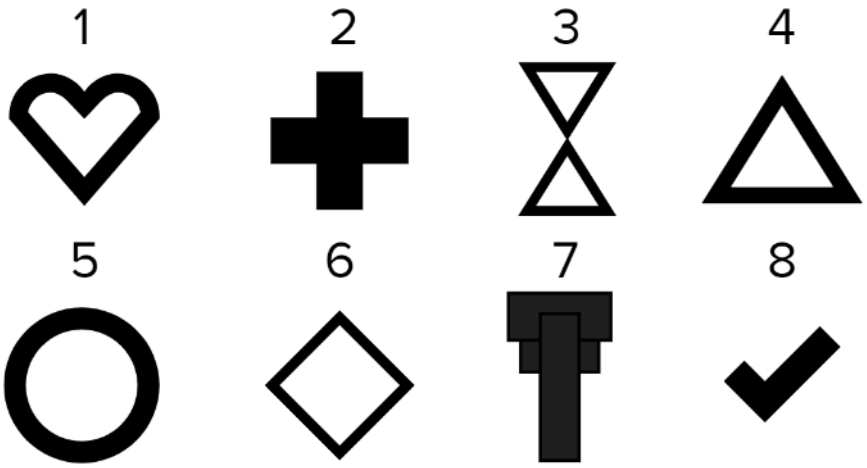

Figura 128: Figuras modelo del segundo programa

Para conseguir que las coordenadas reconocidas en el rostro del usuario puedan ser trasladadas al papel que tendremos por modelo, estableceremos una figura que determinará la relación entre ambos puestos de trabajo. Tanto el primer puesto de trabajo como el segundo puesto de trabajo tendrán el mismo formato y presentarán en la esquina superior izquierda la figura que relacionará ambos (figura 129).

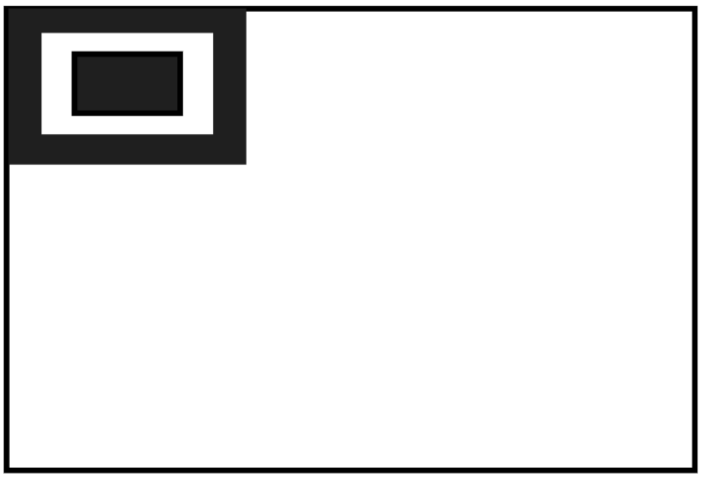

Figura 129: Modelo de hoja para segundo programa

Explicando más a profundidad lo expresado anteriormente, el primer puesto de trabajo tendrá una plantilla transparente, con la forma presentada en la figura 129, del tamaño de la hoja que se

encuentra en el segundo puesto de trabajo. De esta manera aseguraremos que las distancias y coordenadas aplicadas para el mapeo del rostro son correctas. Adicionalmente, definiremos un modelo de "fixed point" sobre la figura característica en el segundo puesto de trabajo para establecer el punto de referencia de los trazos a ser dibujados.

4.3 Programa

A fin de lograr que el robot dibuje una aproximación geométrica del rostro del usuario, necesitaremos posicionar algunos referentes en lugares clave del rostro para lograr preservar sus coordenadas y trasladarlas a papel. Para ello se definirán dos posiciones de trabajo. La primera posición de trabajo será en la que se analice el rostro del usuario guardando las coordenadas de los puntos clave. El segundo puesto de trabajo será la hoja en la que se dibujarán los trazos. Esto se podrá ver ilustrado en la figura 130.

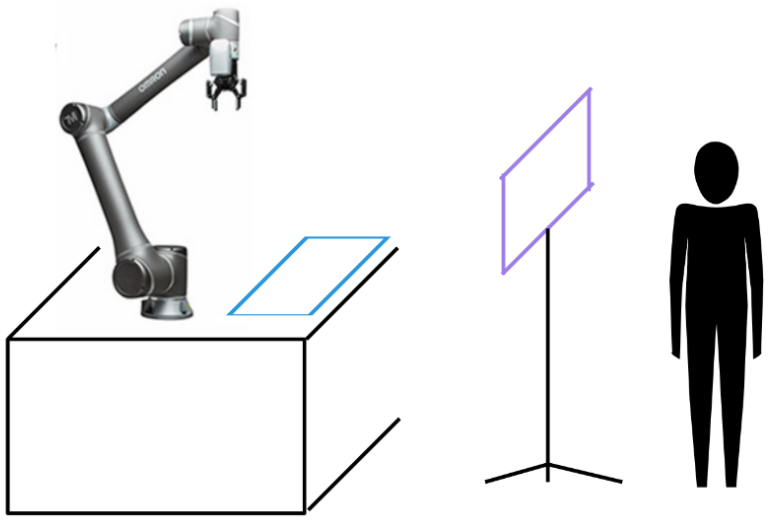

Figura 130: Puestos de trabajo del segundo programa

Nota: podemos sobresaltar que el primer puesto de trabajo está delimitado por las líneas de color morado, mientras que, el segundo puesto de trabajo estará delimitado por las líneas de color azul.

Si bien el código tendrá dos puestos de trabajo, es importante explicar que el robot dibujará el rostro del usuario en dos partes. Primeramente, el robot dibujará el contorno del rostro con las distancias y coordenadas relevantes al usuario. En segundo lugar, se volverá al primer puesto de trabajo para conseguir las distancias concertantes para luego realizar los trazos del interior del rostro. El funcionamiento del programa podrá verse ilustrado por las figuras 131 y 132.

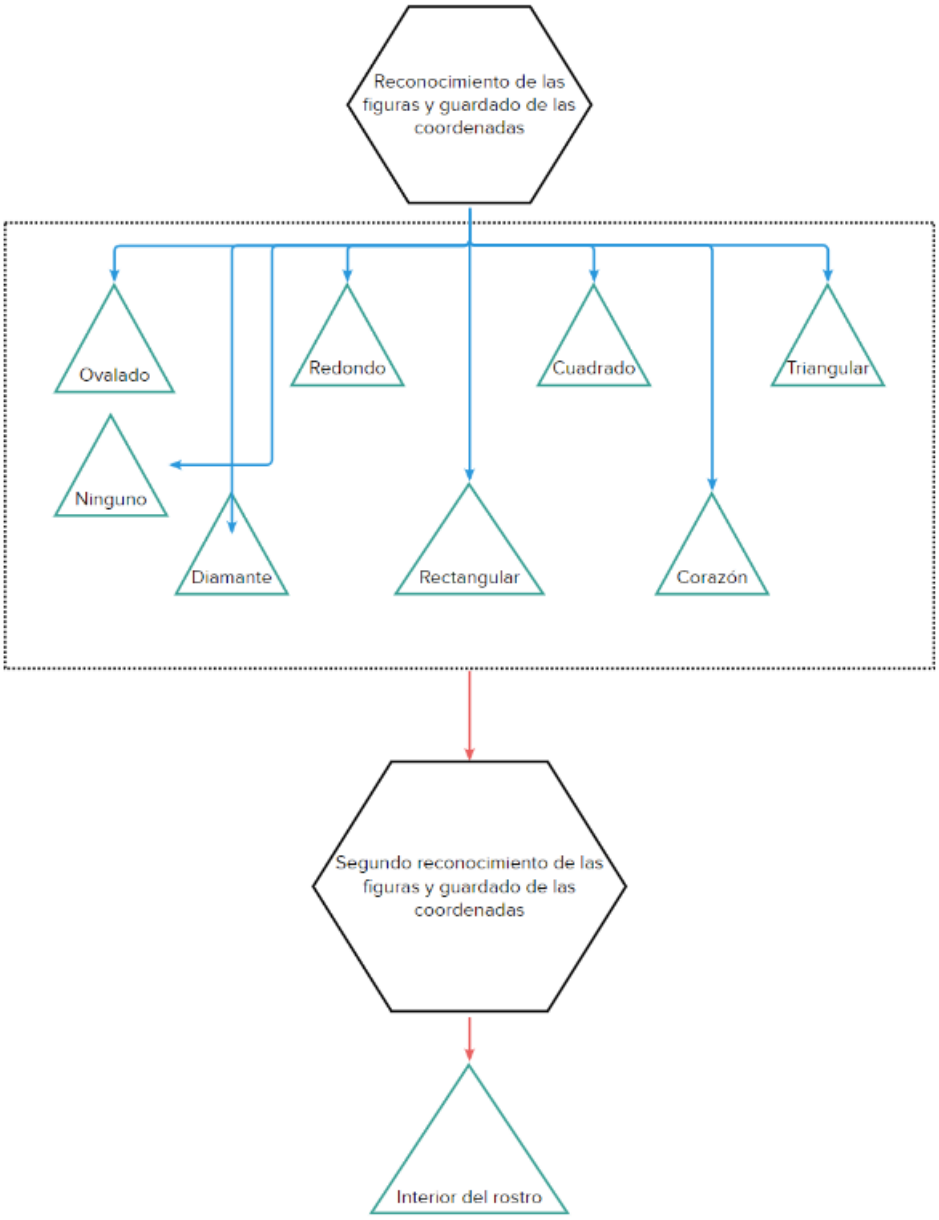

Figura 131: Diagrama de flujo del segundo programa

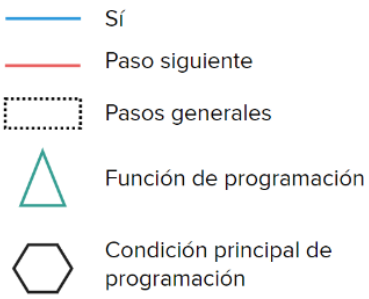

Figura 132: Leyenda del diagrama de flujo del segundo programa

4.3.1 Main

En la página principal de programación del código se identificará el tipo de rostro con el que estaremos trabajando y se realizarán diferentes funciones específicas para cada caso. Para poder definir de manera correcta el caso en el que nos encontramos tenemos que definir la diferencia entre los diferentes tipos de rostros. Para ello, usaremos la distancia entre los puntos 3 y 5 de la figura 126 como determinante. Tomando la figura 125 como referente de proporcionalidad tenemos las siguientes medidas para las distancias entre 3 y 5:

- Corazón: 1,25 cm
- Rectangular: 2,81 cm
- Triangular: 4,69 cm
- Ovalada: 5,6 cm
- Redondo: 6,25 cm
- Cuadrado: 7,2 cm
- Diamante: 7,81 cm

Para calcular las medidas previamente presentadas tomamos los rostros presentados en la figura 125 y desarrollamos una regla de proporcionalidad con la hoja en la cual realizaremos el dibujo (que es del mismo tamaño que la plantilla transparente por la cual se analizará el rostro del usuario). Asumiendo que podemos dibujar a lo largo de la hoja, asumimos que la distancia entre los puntos 6 y 5 puede llegar a 20cm. Al medir dicha distancia en la figura 125 vemos que esta se asemeja a 6,4cm. Por lo tanto, una vez medida la distancia entre los puntos 3 y 5 de la figura 125 para cada caso transformamos mencionada distancia usando la siguiente ecuación:

$$x = d * \frac{20}{6.4}$$

Siendo x la distancia a tener en cuenta a la hora de la programación y d la distancia entre los puntos 3 y 5 de la figura 125.

Como las medidas presentadas previamente son aproximaciones, definiremos intervalos para definir en qué tipo de rostro nos encontramos. Para definir estos intervalos, tomamos la diferencia entre las distancias y las dividimos en dos. Para ilustrar lo mencionado, mostramos el intervalo para definir el rostro cuadrado.

$$\frac{7,2 - 6,25}{2} < x \leq \frac{7,81 - 7,2}{2}$$

Cabe a recalcar la posibilidad de que nos encontremos en un caso que no se vea definido por las medidas e intervalos definidos previamente. Es por esto por lo que definiremos un caso extra en el que no se reconozca correctamente el rostro del usuario. En dicho caso, a la hora de dibujar el contorno del rostro el robot se moverá de manera directa entre los diferentes puntos.

Una vez definidos los casos a ser considerados y las condiciones que los separan, programamos las condiciones necesarias para derivar cada caso a su función correspondiente.

4.3.2 Funciones

En este programa, las funciones se realizarán de manera paralela teniendo como principal diferencia los puntos intermedios que se tomarán para realizar las diferentes formas características de los diferentes rostros. Explicando más profundamente lo nombrado, nos aproximaremos a la forma deseada a través de puntos intermedios establecidos de tal manera que al unirlos nos encontremos en una aproximación adecuada de la figura deseada. Esto se podrá ver ilustrado en la figura 133.

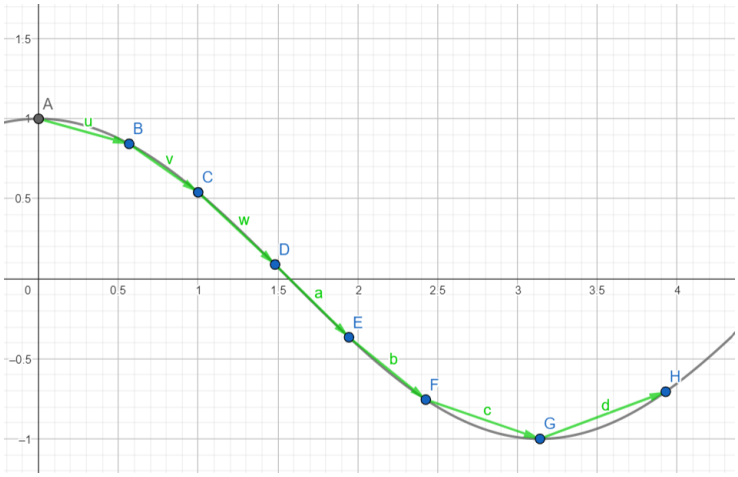

Figura 133: Aproximación de una función mediante puntos intermedios

En cada caso nos aproximaremos al trazo deseado a través de puntos intermedios. Estos puntos variarán dependiendo del caso en el que nos encontremos. No obstante, si bien estos puntos cambian dependiendo del caso, estos estarán relacionados bajo la misma regla. Al analizar el modelo de cerca, dividimos la distancia en x e y entre la primera y quinta figura. Una vez divididas las distancias, se encuentra la proporción en cada eje y se aplica a las proporciones y distancias medidas por la cámara.

A continuación, definimos el número de puntos a ser aplicados por caso y los establecemos de la manera expuesta en las figuras 134 a 140.

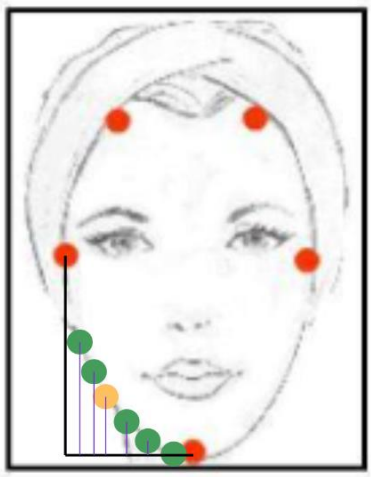

Figura 134: Puntos intermedios del rostro ovalado

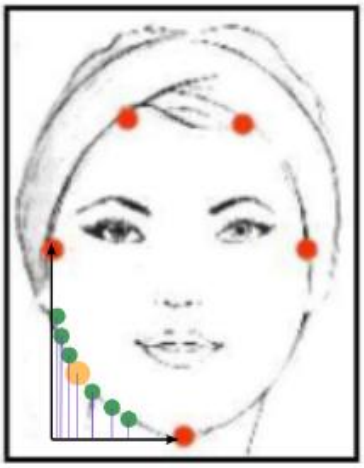

Figura 135: Puntos intermedios del rostro redondo

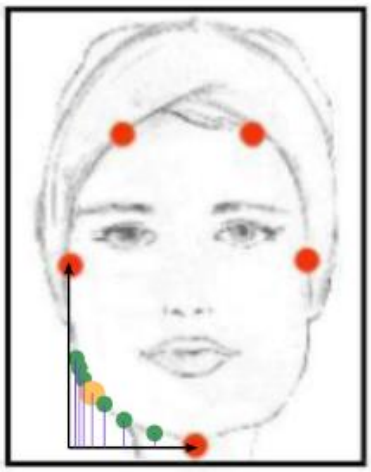

Figura 136: Puntos intermedios del rostro cuadrado

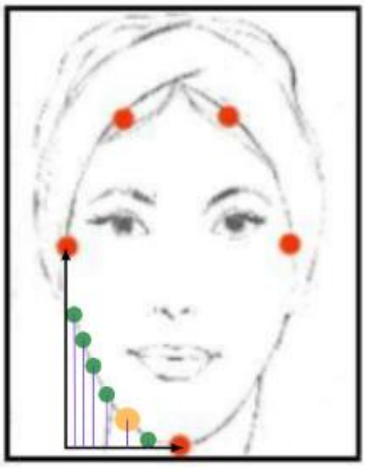

Figura 137: Puntos intermedios del rostro rectangular

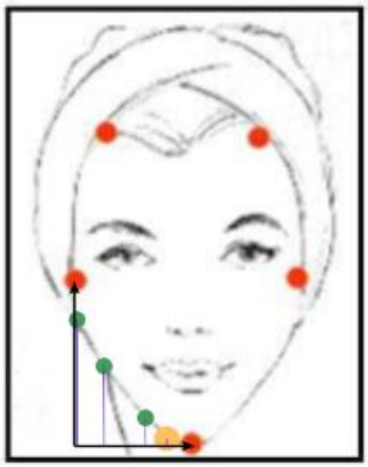

Figura 138: Puntos intermedios del rostro con forma de corazón

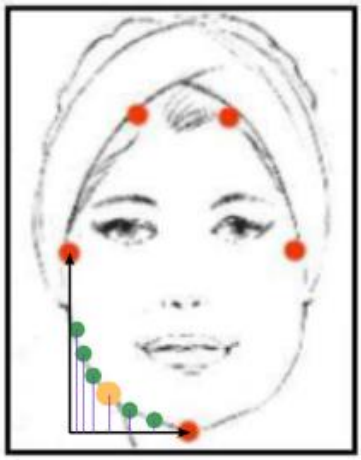

Figura 139: Puntos intermedios del rostro triangular

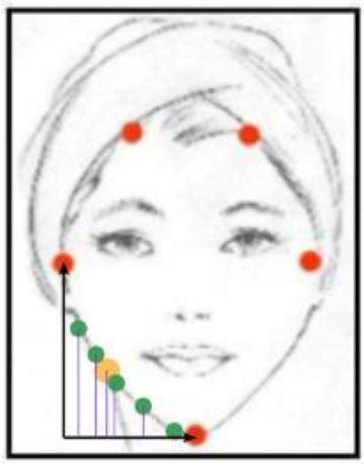

Figura 140: Puntos intermedios del rostro con forma de diamante

Cada una de las funciones seguirá las proporciones y los puntos intermedios referentes al tipo de rostro base. Es importante notar que los puntos intermedios seleccionados para cada rostro se

aplicarán para ambos lados del rostro teniendo en cuenta las diferentes distancias y coordenadas de los puntos referencia. Explayando lo escrito, para poder aplicar los puntos intermedios de manera adecuada, se tendrá en cuenta en todo momento la distancia entre los puntos que nos encontremos y las proporciones serán aplicadas a dicha distancia. Más explícitamente, si nos encontramos entre los puntos 1 y 3 (ver figura 126) aplicaremos las proporciones encontradas en dicho intervalo y estas últimas serán aplicadas a la distancia medida entre ambos puntos. De esta forma podremos asegurarnos de tener una aproximación lo más cercana a las distancias y proporciones reales del rostro humano.

4.3.3 Interior del rostro

Para diseñar la programación del interior del rostro nos refugiaremos en la geometría (figuras 141-143). Como realizado en la programación de las funciones previas nos basaremos en proporciones que luego serán aplicadas a las distancias tomadas del rostro del usuario. La programación que se llevará a cabo en este apartado será la misma para todos los rostros.

Figura 141: Geometría básica del ojo humano

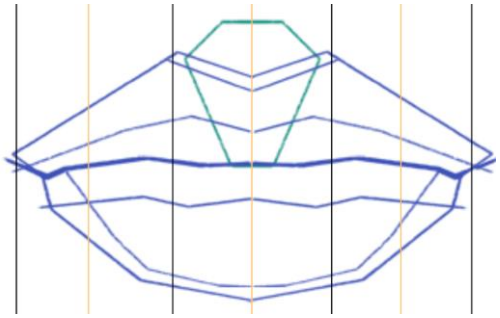

Figura 142: Geometría básica de los labios humanos

Figura 143: Geometría básica de la nariz humana

Como observado en las figuras previamente mostradas, la geometría escogida para realizar los trazos se aproxima bastante a la figura original. No obstante, esta sigue siendo una aproximación que introducirá errores a la hora de compararlo con la figura original. Adicionalmente, como se puede observar, los puntos intermedios escogidos son reducidos por lo cual favorecerán las diferencias entre los trazos y la realidad.

4.4 Errores

Puesto que la programación del código recae principalmente en la calibración y funcionamiento de la cámara es natural que el primer error sea el generado por la calidad de ambas. La medición generada por la cámara contendrá desde el principio un error que estará directamente relacionado con el error en la calibración. Este tipo de error será el que limite la exactitud de los trazos realizados por el robot.

Adicionalmente, cabe a recalcar que el estilo de programación seleccionado presenta, a su vez, errores a la hora de comparar escrupulosamente los trazos con el rostro real. Esto se debe a que, como se ha mencionado previamente, los puntos intermedios son numéricamente bajos y escogidos arbitrariamente.

Ambos fenómenos contribuyen a los errores que se pueden presentar una vez el programa sea puesto a prueba. No obstante, también se pueden presentar errores que sean externos a la programación del robot. Dentro de estos errores podemos recalcar dos. El primero, se ve relacionado con el agarre que posee la garra sobre el instrumento que nos permita realizar los trazos. Esto se debe a que, al ser la garra rectangular el agarre que esta pueda presentar sobre el instrumento no es distribuida sobre todos los ejes lo que presenta deslices y errores a la hora de realizar los trazos. El segundo, se da puesto que el puesto de trabajo en el que se encuentra la hoja no es totalmente plano, ya que, presenta una elevación que dificulta el movimiento del robot en un mismo plano (figura 144).

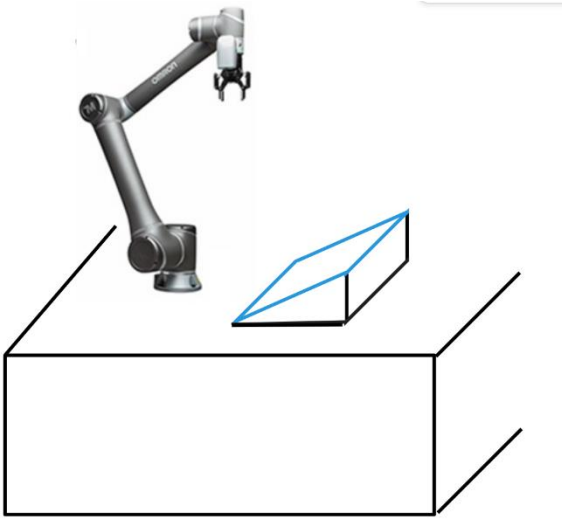

Figura 144: Elevación del primer puesto de trabajo

4.5 Conclusiones del programa

Si bien el programa explicado reside más en la explicación teórica, podemos claramente visualizar su funcionamiento ya que este programa prospera en su pura simplicidad.

Aunque el programa explicado presenta una simplicidad de diseño y desarrollo, éste muestra a su vez cómo, incluso con recursos diseñados para aplicaciones específicas estos se pueden redireccionar y destinar a aplicaciones totalmente distintas.

5.CONCLUSIONES GENERALES

En conclusión, los objetivos trazados fueron alcanzados después de realizar un análisis del funcionamiento del brazo robótico y de sus periféricos, comprendiendo de manera práctica la interfaz del sistema. La planificación y creación de los códigos básicos de programación del robot culminaron en exitosos resultados. De la misma manera, se logró analizar y enfrentar los errores de funcionamiento presentes en el desempeño de la programación del robot.

Adicionalmente, es importante destacar que las aplicaciones derivadas de la comunicación entre el brazo robótico y la cámara con visión artificial son varias. El haber empujado dicha comunicación a niveles de programación robustos, nos muestra claramente las posibilidades futuras de desarrollo en este ámbito. Paralelamente, los errores encontrados durante el proceso nos proporcionan una perspectiva valiosa para futuros trabajos.

Por último, podemos recalcar la importancia de la prueba y ejecución de los programas que colaboran con la detección y solución de problemas que pueden pasar inadvertidos durante el desarrollo teórico de la programación.

6.PRESUPUESTO

En este apartado nos enfocaremos en presentar un presupuesto estimado para la realización de la programación algorítmica de un brazo robótico. Tanto el presupuesto como los costes se han dividido de la siguiente manera.

1. Mano de obra estudiantil: se ha estimado un coste total de 4.500€ para la contratación de un estudiante. Dicha estimación se basa tomando en cuenta que el estudiante trabaja por 15€/h durante 300 horas.
2. Mano de obra de un supervisor: se ha previsto que el coste total sería de 7.500€ para la incorporación de personal que supervise el trabajo del estudiante. Esta estimación se apoya bajo la asunción que el supervisor trabaja por la mitad del tiempo que el estudiante cobrando 50€/h.
3. Amortización del robot: apoyándose en el método lineal de amortización con una tasa de depreciación del 10%, el gasto de amortización del robot para los 4 meses en los que fue utilizado se valora en 1000€.

Presupuesto de mano de obra

Tarea	Duración (horas)	Precio (€)	
		Unitario	Total
Estudiante	300	15	4500
Supervisor	150	50	7500
Total mano obra			12000

Tabla 7: Presupuesto Mano obra

Presupuesto de equipamiento

Equipamiento	Cuota adquisición (€)	Tiempo amortización (años)	Cuota amortización mensual (€)	Tiempo de uso (mes)	Amortización (€)
Máquina	30000	10	250	4	1000
Total equipamiento					1000

Tabla 8: Presupuesto equipamiento

En total, el presupuesto previsto para la realización del proyecto es equivalente a 13.000€. Es importante remarcar que el presupuesto presentado es totalmente estimativo por lo que este puede variar en función a diferentes factores. Es relevante subrayar que se ha tratado de realizar asunciones reales con tal de garantizar la fiabilidad del presupuesto económico del proyecto.

7.BIBLIOGRAFÍA

[1] Ejemplo de movimiento del robot - https://www.youtube.com/watch?v=A6dy7Ro0QW4

[2] Ejemplo de movimiento del robot - https://www.youtube.com/watch?v=kuweSnXQqX0

[3] Ejemplo de movimiento del robot - https://www.youtube.com/watch?v=ItBV7D9-k-I

[4] Ejemplo de como localizar un objeto - https://www.youtube.com/watch?v=0-jNS8pirdQ

[5] Estudio de formas de cara - https://www.google.com/search?q=diferentes+formas+de+caras&rlz=1C1GCEA_enES926ES926&sxsrf=ALiCzsYPNkIWj6iXsB4VfgYOK_JousRrKw:1664545896407&source=lnms&tbm=isch&sa=X&ved

=2ahUKEwjHivjW1Lz6AhXWgf0HHUKXD6YQ_AUoAXoECAIQAw&biw=1536&bih=714&dpr=1.25#imgrc=2RQ1__aV1GbPGM

[6] Canon - https://blogsaverroes.juntadeandalucia.es/jvaraujo/epv4/analisis-de-formas/el-canon-en-el-rostro-humano/

[7] 7 tipos de rostros - https://www.hogarmania.com/belleza/estetica/rostro/cual-forma-rostro-24719.html

[8] Evolución histórica de la robótica industrial - https://www.edsrobotics.com/blog/evolucion-robotica-industrial/

[9] Evolución histórica de la robótica industrial - https://www.empresaactual.com/evolucion-y-tendencias-de-la-robotica-industrial/#:~:text=Evoluci%C3%B3n%20de%20la%20rob%C3%B3tica&text=La%20historia%20de%20la%20rob%C3%B3tica,y%20realizaban%20un%20movimiento%20repetitivo.
https://www.lavanguardia.com/vida/junior-report/20220610/8330874/historia-robotica.html#:~:text=A%20principios%20de%201960%2C%20la,montaje%20automatizadas%20para%20fabricar%20autom%C3%B3viles.
https://www.logicbus.com.mx/blog/puma-el-pionero-en-robots-de-ensamblaje/

[10] Inteligencia artificial – Oracle Cloud - https://www.oracle.com/mx/artificial-intelligence/what-is-ai/

[11] Inteligencia artificial – SAS - https://www.sas.com/es_cl/insights/analytics/what-is-artificial-intelligence.html

[12] Inteligencia artificial – TechTarget - https://www.techtarget.com/searchenterpriseai/definition/AI-Artificial-Intelligence