IGUS Six-Joint Robot – Tres en Raya – Documentation

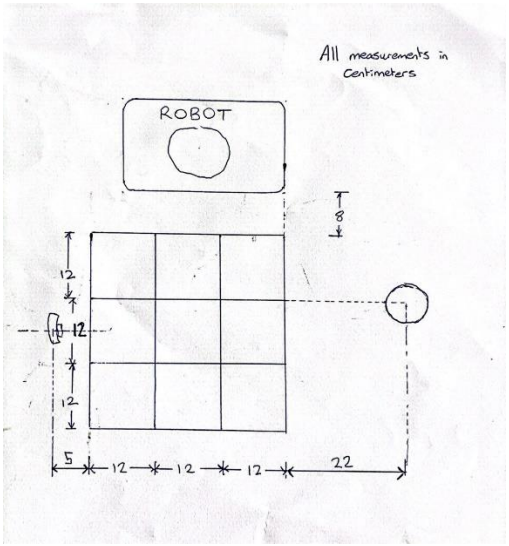
Henry Beare and Marcos Nielsen

Introduction

This document will explain the code that was written for the Tres en Raya project, completed during July and August of 2024. The code can be found on github: https://github.com/hb677/igusrobot-tresenraya

Game Space and Equipment

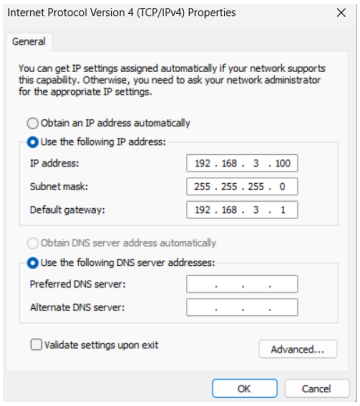
The aim was to use an IGUS ReBeL cobot 6 to physically play tres en raya against a human opponent. The equipment must be set up as shown:



Setting Up the Ethernet Connection

To set up the ethernet connection with the IGUS robot first an ethernet cable must be connected between the robot and the computer. Then the following section in the 'Control Panel' must be accessed: 'Control Panel\Network and Internet\Network and Sharing Center'

Then select the ethernet connection, select Properties, Internet Protocol Version 4 (TCP/IPv4), and Properties again. Then add the following information:



Code

There are two different sets of code: ‘tresenraya.py’ and ‘tresenraya-conboton.py’. They are near identical, the only difference is that ‘tresenraya-conboton.py’ has some changes to allow it to be used with a Raspberry Pi and a physical button.

tresenraya.py

‘tresenraya.py’ uses functions from two other files ‘igus_robot.py’ and ‘board_analysis.py’. It uses the functions from these files to initialise, and play the game.

igus_robot.py

This contains the function which allows for messages to be sent to the IGUS robot over the ethernet connection. More details can be found at the following link: https://wiki.cpr-robots.com/index.php/CRI_Ethernet_Interface

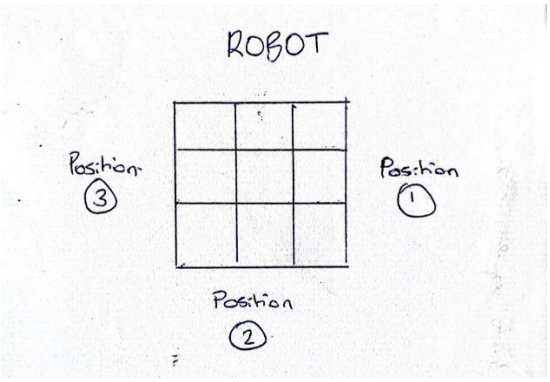
board_analysis.py

This contains the function, ‘check_player_move’ which uses the camera to take a picture of the board and uses the OpenCV library to analyse where pieces have been placed.

It begins by finding the co-ordinates of the four corners of the board in clockwise order: top left, top right, bottom right, bottom left. It does this by applying the ‘sort_clockwise’ function on the output of the ‘get_bounding_box’ function. These are both found in the ‘find_board.py’ file. The ‘get_bounding_box' function finds the contours on the inputted image and returns the second largest contour. If the camera is positioned correctly the second largest contour will always be the contour which defines the board. This is because the largest contour is always found around the entire edge of the image.

Then it uses the ‘perspective_change_matrix’ and ‘change_perspective’ functions to flatten the image as initially the image is skewed.

Next, depending on the camera_position the image is rotated.



Next, the ‘slice_and_return’ function is used to slice the image of the game board into its nine constituent squares.

Then we iterate through the ‘board_list’ and only when a space on the board is empty do we use the ‘is_empty’ function to check if the square is still empty. The robot already

knows when it places a circle so therefore it only has to check for crosses. Therefore we know if a square that was previously empty is now not empty it must contain a cross.

The 'is_empty' function is stored in the 'shape_analysis_v3.py' file. It uses the 'get_contours_colour_masking' function to find the contour of the piece on the square. It does this by filtering out all colours in the image except the colour of the pieces. Our pieces were green and so the HSV ranges we used were 35-100 for HUE, 60-255 for SATURATION, and 110-255 for VALUE. More information on the HSV scale can be found here: https://en.wikipedia.org/wiki/HSL_and_HSV . If a contour found in the image takes up over 8% of the total area then we know it cannot just be noise and must be one of the pieces. Therefore this square is not empty.

Initialisation

The code begins by setting the camera port and the camera position. The camera port is a value which allows the code to know which camera to use. If the camera for tres en raya is the only camera connected then this value will be 0. But for example if the computer already had a built in webcam and the camera is then connected on top of this then the camera port value may be 1, for example.

Then the robot is connected to, reset, and enabled.

Gameplay

The game plays in a loop which will be broken if the player chooses not to play again once the game is finished.

Initially the list which represents the board is reset, the number of pieces left is reset to five, and the code asks for who is going to play first.

When it is the players turn, the code only has to wait as it does not need to perform any actions.

When it is the computers turn, it uses the camera to take an image of the current game space. It then uses the 'check_player_move' function which was mentioned previously to update the list which represents the board. It then checks if, after the last move, the player has won. If not, it then uses the 'get_computer_move' function to decide what the best move for the computer to make is.

The 'get_computer_move' function uses the following algorithm to choose a move:

1. Is there a move that the computer can make to win? If yes then make that move. If no then move to next step.
2. Is there a move that the player can make on their next turn to win? If yes then make that move in order to block the player. If no then move to next step.
3. Is there a free corner? If yes then move to a free corner. If no then move to next step.
4. Is the middle free? If yes then move to the middle. If no then move to next step.
5. Is there a free side square? If yes then move to a free side square. If no then move to next step.
6. No possible move and therefore a draw has been reached.

Now that we know which move to make, the code executes the 'make_move' function to change the state of the list which represents the board. It then executes the 'pickup_and_place' function so that the robot physically makes the move on the board.

The ‘pickup_and_place’ function first loads the relevant pickup XML file and executes it. It then loads the relevant place XML file and executes it.

This continues, cycling between player and computer moves, until either one wins or a draw is reached.

XML Movement Files

The movements that the robot must make are stored in XML files that we edited using the ‘igus Robot Control V14’ software. The fifteen movements must be stored in the following file location: C:\iRC-igusRobotControl-V14\Data\Programs