Robotics Project: Robomaster Parkour

Jonatan Bella, Fabian Gobet, Tobias Erbacher — *Università della Svizzera italiana*

***Abstract*—In this project, we develop a Python-based PID controller for the DJI RoboMaster EP (hereafter abbreviated by ”RoboMaster”), equipped with a camera, which we will use as input to Computer Vision algorithms to navigate through a simulated parkour of obstacles, as well as a robot arm with a gripper attached to its end, which we will use to grab objects autonomously.**

I. Introduction

WHEN a robot has to find a path through unknown terrain it can be difficult at times to identify the best available path to follow. In this project we focus on developing computer vision algorithms that identify predetermined gates where the robot can pass through autonomously, move closer to and thereafter through the gates safely without coming into contact with them.

May 30, 2024

II. Problem Description

A. The Goal

We aim to demonstrate that we achieved grabbing objects and navigating a parkour, relying solely on simple geometry and computer vision techniques with only a camera feedback. This can have many possible applications in industrial setups where a robot might need to be able to pick up products/supplies and be able to transport them through a chaotic and unforeseeable environment.

B. The RoboMaster

The RoboMaster is a customizable mobile robot platform, mounted on four mecanum wheels[1] which allow for omnidirectional movement to the front, back, sideways and in-place-rotation, equipped with a RGB camera that can return live view images at a rate up to 30 fps. Moreover, it has a robotic arm with a gripper attached at the end which we will make use of experimentally at a later stage.[2] The robot possesses other sensors too, however those presented are the ones we will be using in this project. In figure 1 the digital model is depicted.

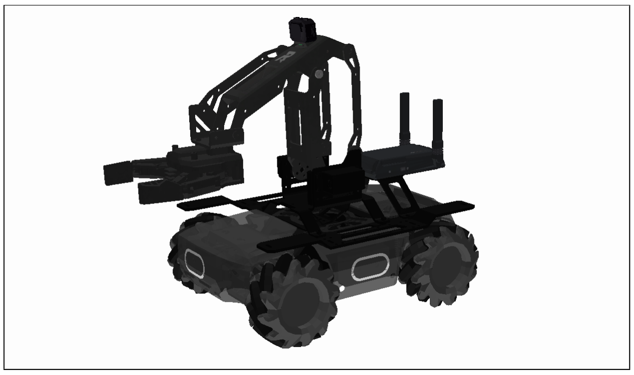

Fig. 1: The RoboMaster Model.

C. The Parkour

The Parkour consists of gates, the model of which can be seen in figure 2. As seen from the perspective of the robot, while passing through the gate the green marker will always be on the left and the red one will always be on the right. We can use this to identify in which direction markers are rotated and determine in which way the RoboMaster needs to move. From the back side the markers are covered in white so that we will never run into the danger of getting stuck at one gate where the robot moves back and forth through the same gate. Moreover, the markers have a gap of 60 centimeters between them so that there is a little bit of wiggle room in case the RoboMaster does not hit the exact center of the gate to move through it. Furthermore, the colors used in RGB format are (66, 255, 0) for green and (255, 0, 31) for red, both defined in percent so there are rounding errors. The shape is created from a vertical cylinder and the colors are only visible in one quarter of the lateral surface area so that we can use the yes/no answer to whether a marker is visible as a first estimate for where we are in relation to it.

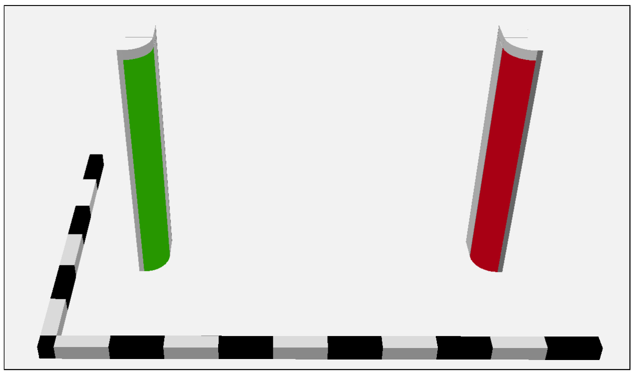

Fig. 2: The model of the gates. The size indicators are spaced in 10cm intervals. Note that the RoboMaster model has a width of slightly less than 25cm in the simulation.

The parkour is then created by placing the gates sequentially after each other in a scene where we know that after successfully passing through one gate, the next one is already visible. The parkour which we will use for demonstration purposes can be seen in figure 6.

III. Solving the Problem

A. The RoboMaster controller architecture

We set up the RoboMaster controller as a state machine of in total 13 states. States 0 to 4 deal with gripping an object, these will be explained in section IV. States 5 to 12 relate to the gate identification and subsequent corresponding movement:

- States 0 - 4:
  Search an object, grab the object, and initiate the parkour traversal. For more details see section IV.

- State 5 – SEARCH:
  The initial state of the robot, searches for any gate marker whose size is above a certain threshold. Until a gate is identified, the robot rotates in place. Once a gate has been discovered, we transition to state 6.

- State 6 – MOVE_CLOSER:
  Until the threshold of desired apparent marker size in either the green or the red marker has been reached for the identified gate, we approach the gate in a way that is dictated by PID controllers. The input to those is (a) how far off is the center of the visible marker compared to the desired apparent horizontal position in the camera, and (b) how large is the apparent marker size in relation to the threshold size. Once the threshold of desired apparent marker size as just described is surpassed, we transition to state 7.

- State 7 – OCR_INIT:
  This state is a decision maker state. The robot comes to a standstill and checks whether both apparent marker sizes, i.e. the red and the green one, are above the desired threshold. If this is the case, then we transition to state 10, else we transition to state 8.

- State 8 – OCR_SEARCH:
  If we transition to this state, then likely the robot sees only one of the markers. To find the second one, it rotates in place until it sees the other's area above a certain area threshold. The direction of rotation then depends on which marker the robot sees. If it is the red marker (i.e. the one on the right), then it will rotate from the perspective of the camera to the left and vice versa. If the RoboMaster cannot find the second marker, then it transitions to state 9. If it sees both markers sufficiently large, then we transition to state 10.

- State 9 – OCR_1:
  In this state we move backwards from the currently identified single marker with the goal of finding the second, since in the previous state rotating in place did not help finding the second one. To this end we first check whether we have previously passed through and recorded any gate, if this is the case, then we use that information to avoid bumping into the last one while moving backwards, otherwise we simply move backwards controlled by a PID. Once we have found the second marker, we transition to state 10.

- State 10 – OCR_2:
  In this state, we align ourselves with the gate so that the robot looks perpendicularly onto the plane of the markers in order to prepare for passing through the gate's center. To achieve this, we estimate the center between both gate markers in the camera frame and until the perceived center is within a small threshold of the camera center, we turn the robot. Once the alignment is complete, we transition to state 11.

- State 11 – PASS_THROUGH:
  The pass-through is governed by a timer that checks for how long the robot has been moving straight through the gate, and after 5 seconds it stops. While moving through the gate, the robot continuously evaluates the deviation of the gate's center from the camera frame's center and corrects its trajectory if needed. After a successful pass-through, the gate is added to the obstacle map and the RoboMaster transitions to state 12.

- State 12 – AFTER_SEARCH:
  This state serves to restart the search for the next gate by stopping and rotating in place to find the next gate marker. If any sufficiently large marker is found, then we return to state 6.

Now that we have shown between which states the robot can switch, we can look into the individual techniques in more detail.

B. Identifying the gates

In the controller we create a subscription that retrieves that camera frame from the robot every 0.1 seconds. The images are not necessarily evaluated at the same rate but whenever for example the new markers are identified we always choose the latest available image whenever we want to update the marker position for controlling the movements. In addition, we took into account this time variability in our PID implementation by saving the previous computation time step such that the derivative is computed in the corresponding time interval of the calculated error. We carefully chose the design of the gates to make our life simpler with the gate identification by giving them some clear color distinction that is not easy to mess with in the light-simulation of CoppeliaSim. Moreover, the color scheme of the gates also gives us a rough spatial idea of where the robot is located relative to the gate. We tested out two different approaches to identify a gate, one is an exact method implemented using library functions and the other one is an approximation algorithm that we came up with by ourselves. Depending on the choice of frame, the approximation achieves errors up to less than 5% in the area while executing faster.
The first part of the algorithms is the same for both. From the camera frame we create two masks by filtering out every pixel that is not green or red, respectively, so that we get two binary frames represented by boolean pixels, a white pixel means in the original frame the pixel was green or red, respectively, and a black pixel means that it was not. For each of the color generated masks we could have more than one connected component, it could be the case that we have some other gates in frame that present themselves at a further distance. Hence, after primarily filtering each frame and generating the color masks, for each we perform a Depth-First-Search run through in order to get all connected components within each of the these. Our primary goal is to always approach and pass through the closest gate, hence we solely consider pixels that lie within the biggest connected component. The resulting filtered color masks are then continuously stored as a class object and accessed by the remaining function for any decision-like process.

For a proof of concept, gates in this project are specifically designed regarding their colors. We know that Coppelia has shading properties too, hence the first step in this process is to convert each frame to HSI and use specific filtering values to generate two separate masks: A red mask and a green mask. Let each mask of frame M be a binary mask where

$M(i,j) = 1 \Leftrightarrow$ the pixel at position (i,j) belongs to the color of interest and zero otherwise. We define an undirected graph $G = (V,E)$ where each vertex $v \in V$ corresponds to a pixel (i,j) with $M(i,j) = 1$. An edge $e \in E$ exists between vertices (i_1, j_1) and (i_2, j_2) if and only if $\sqrt{(i_1 - i_2)^2 + (j_1 - j_2)^2} = 1$ meaning that each pixel has edges to its 4-neighborhood (up, down, left, right). Then, each of the masks is passed through a pixel-wise DFS algorithm, as previously described, such that it identifies the connected components by running DFS over G. Once identified, the biggest set of pixel coordinates (corresponding to the biggest connected component), is used to generate a boolean mask which accurately represents the target corresponding to pole of the gate, red or green. An example of this process can be seen in figures 3, 4 and 5.

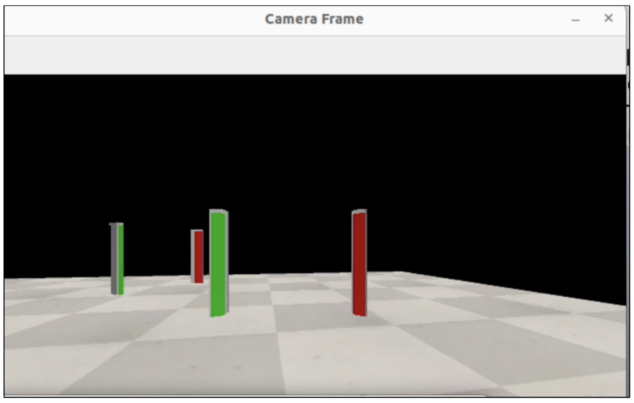

Fig. 3: Multiple gates in frame.

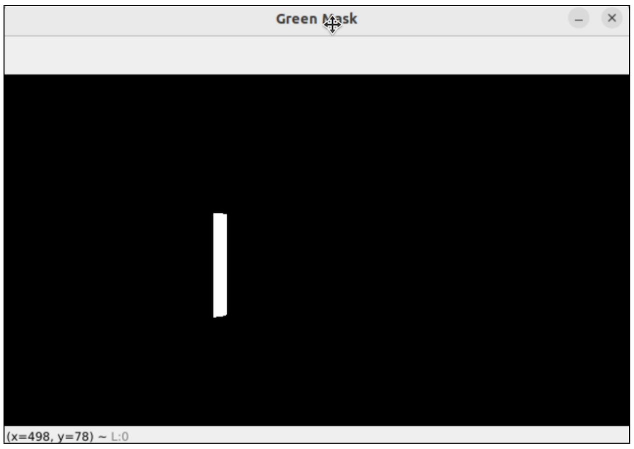

Fig. 4: Correctly filtered pole — green.

C. Navigating the parkour with a PID controller

In each of the states, a different moving mechanism and set of parameters are considered. It is worth noting that most parameters (thresholds) are considered system parameters and can be fine-tuned. State 5 is simply comprised by a rotation in place (along the z-axis) until one of the poles of the gate is identified. To identify a gate means that in the color masks a certain threshold area criterion is met. State 6, the approximation state, is controlled by a PID that compares the area of the identified object to generated linear velocity, and centeredness of the object to generate angular velocity. State 7 evaluates if we are in a position where we see one or two poles of the gate, going to state 8 or 10, respectively. Coming to state 8 means we only identified one pole of the gate. Depending on the color of the pole, an in-place rotation to the

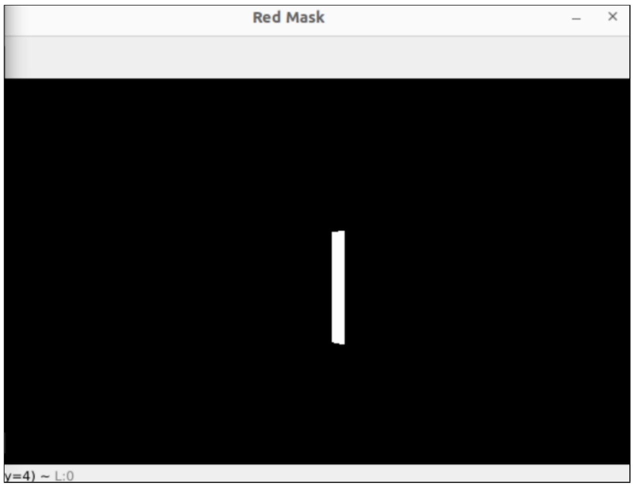

Fig. 5: Correctly filtered pole — red.

opposite direction of the identified pole is made at a constant velocity in an attempt to find the other colored pole. During state 8, if the gap between the already seen pole and the corner of the frame is under a configurable threshold distance, then It means we can effectively only identify one pole and we move to state 9, where we have to perform an object centered rotation around that pole until we can find the other. In this state, the lateral movement (side-ways, in the y-axis) is performed at a constant velocity and the sign is dependent on which color we are currently seeing. The linear velocity is is generated by a PID that considers the area of the seen color, whereas angular velocity is generated by a PID that considers as an error the difference between the intended lateral position of the color pole. From state 9 we must at some point find the second colored pole of the gate, coming to state 10. In this state, we perform an analogous behaviour to state 9, but now the angular velocity is determined through a PID that takes in the difference between the middle distance of the poles and the center of the frame. Furthermore, angular velocity is also determined through a PID that takes into account the absolute differences between the highest points and the lowest points of both colored poles of the gate, whereas linear velocity still depends on an area threshold. Once a position close to perpendicular is achieved, we go to state 11 and the RoboMaster initiates a forward movement at constant velocity for 5s, time which is also a configurable system parameter. Finally, in state 12 the robomaster simply looks left and right at constant rates, with a configurable clock cycle, to search for other gates. (Appendix B) Notice that we created one class with the architecture of the PID that is applied in each case and we instantiate it two times, one for the special case of OCR_2 and one for the rest. OCR_2 movements are more sensitive to error, therefore its parameters are required to be lower. However, for the rest, similar parameters works well and we added a method to reset the error accumulation before each transition. This allows us to keep the code clean.

IV. Additional Functionalities

Our obstacle course is designed to rigorously test the capabilities of the robomaster. The course begins with a shelf featuring a blue square object, which the robomaster must approach and retrieve. Following this, the course includes several gates arranged in a circular pattern without any walls separating them. This configuration is intended to challenge the robomaster's ability to infer and navigate multiple gates efficiently, as well as to assess its proficiency in perpendicular alignment and gate traversal. This setup provides a comprehensive evaluation of our implementations, ensuring robust performance across various scenarios.

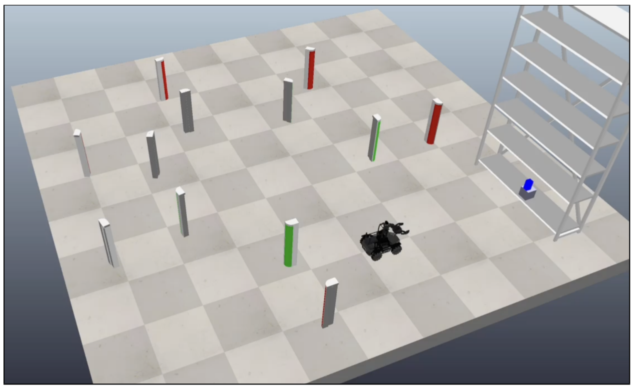
Fig. 6: Obstacle course.

With respect to object picking implementation, the RoboMaster initializes in 0 state where it turns in place until identifying a blue cuboid waiting to be picked. When perceived, it builds a contour around the cuboid and calculates the centroid of it. Therefore, the approach to the target is proportional to the error of the difference between the image frame center and the cuboid contour centroid pixel position. When the area of the cuboid is over a certain tunable threshold, the robot stops and transitions to the following state that consist of approximating the gripper around the cuboid. This is also done by moving the arms diagonally by accelerating over the x and z axis of the robot arms until the target area reaches another tunable threshold, transitioning to the grasp cube state.
During the grasp cube state, the action for grasping the object involves a series of steps orchestrated by the ”grasp_cube” method. It starts by stopping the robot and sending a command to open the gripper. The ”send_gripper_command” method handles the communication with the gripper action server, sending the goal message and setting up callbacks for the response and feedback. The ”goal_response_callback” and ”get_result_callback” methods handle the response and result from the action server, respectively. The ”feedback_callback” method processes the feedback received during the execution and determines the next steps based on the current gripper state. If the gripper is open, it proceeds to close the gripper using the ”close_gripper” method. If the gripper is closed, it proceeds to lift the cube using the ”lift_cube” method, which involves lifting the arm, moving the robot in reverse, stopping the robot, and lowering the arm. Finally, the last state of the cube grabbing implementation is setting up the transition to the previously described implementation for navigation.

V. Further Development

One possible extension to our development with respect to the cuboid grabbing implementation is to use visual servoing. This can be easily done in RoboMaster since we can retrieve the camera’s internal parameters by a topic subscription (in fact, ”/RoboMaster/camera/camera_info” provides us with the calibration matrix K and the projective matrix P.

VI. Conclusion

Our primary objective was to have a RoboMaster navigate through a series of gates, successfully following a predefined course from start to finish. To achieve this, we decomposed the problem into smaller tasks, resulting in the development of a state machine. Through this methodical approach, we accomplished all the planned milestones. One of the more challenging aspects was the object-centered rotation algorithms, which were highly sensitive to parameter thresholds. This necessitated extensive fine-tuning of system parameters and the PID controller to achieve optimal performance. Having completed the initial objectives, we extended the capabilities of the RoboMaster by implementing functionalities that enabled it to pick up and carry an object through the obstacle course. This additional feature transformed our simulation into a proof of concept, demonstrating the potential for a robot like the RoboMaster to autonomously handle and transport objects along a defined path. The success of this project highlights the value of such robotic systems in applications where handling hazardous materials is required, providing a safer alternative to human intervention.

References

[1] Wikipedia. *Mecanum Wheel.* URL: https://en.wikipedia.org/wiki/Mecanum_wheel.
[2] Da-Jiang Innovations Science and Technology Co. Ltd. *Robomaster EP Specs.* URL: https://www.dji.com/ch/robomaster-ep/specs.

Appendix A
Code Note

For testing purposes, we include three different versions of the code in the upload. There is one with only the parkour navigation, one with only the gripper functionality, and then there is the version which we are referring to in this report where both codes are merged. Further explanation can be found on the readme file of the submitted ROS package.

APPENDIX B
GRAPH REPRESENTATION OF THE MAIN STATE MACHINE

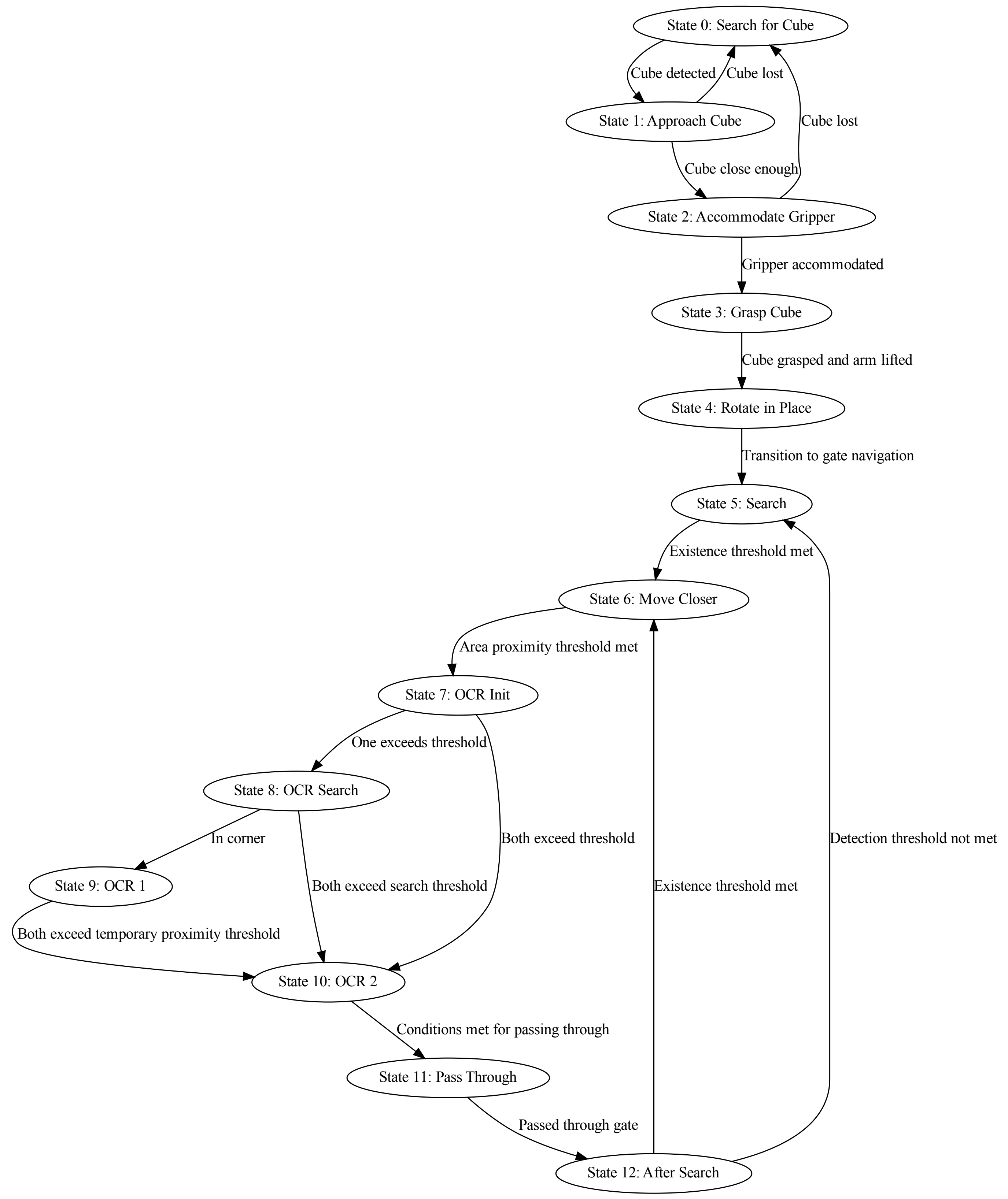

Fig. 7: Graph representation of our implemented state machine where each edge is the result of each corresponding method.