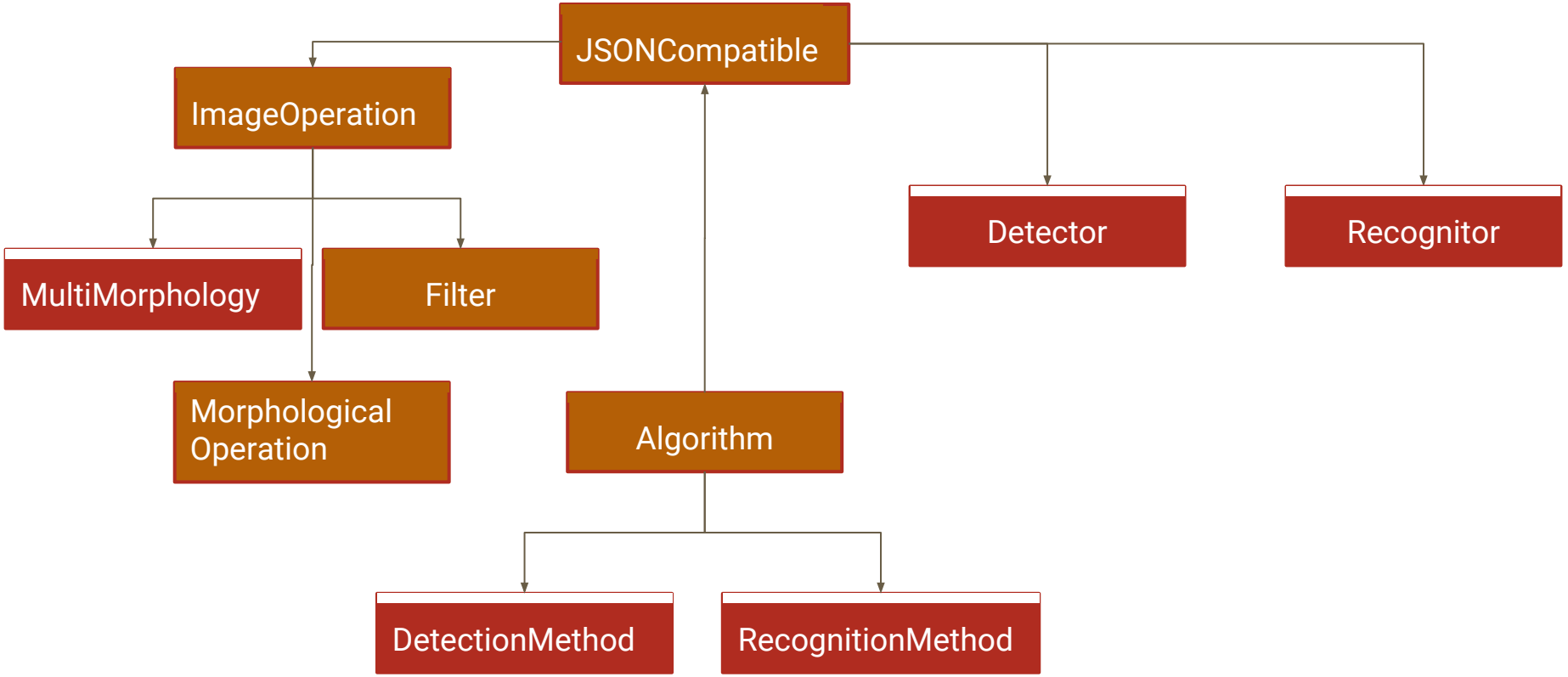
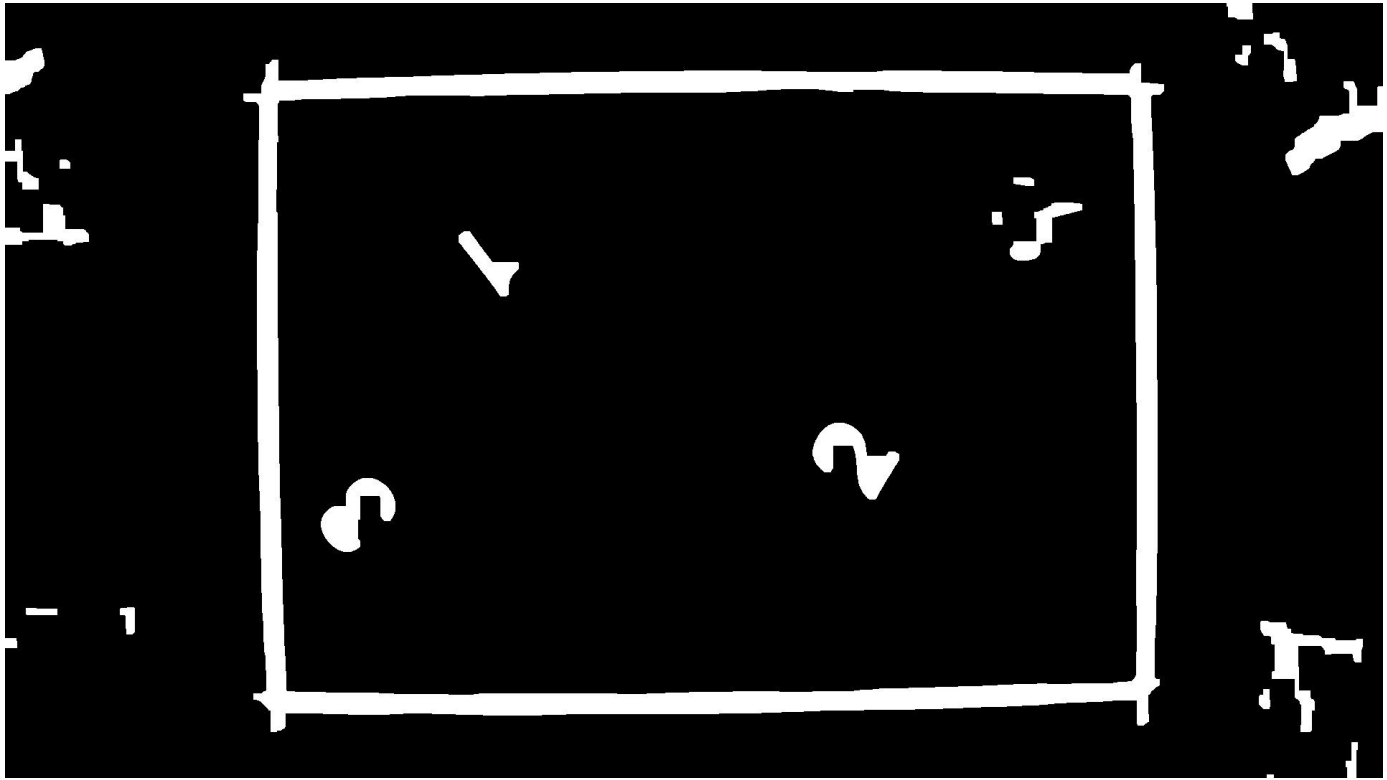
Laboratory of Applied Robotic Final Presentation

Sergio Catalano,
Simone Zamboni
25/01/2019

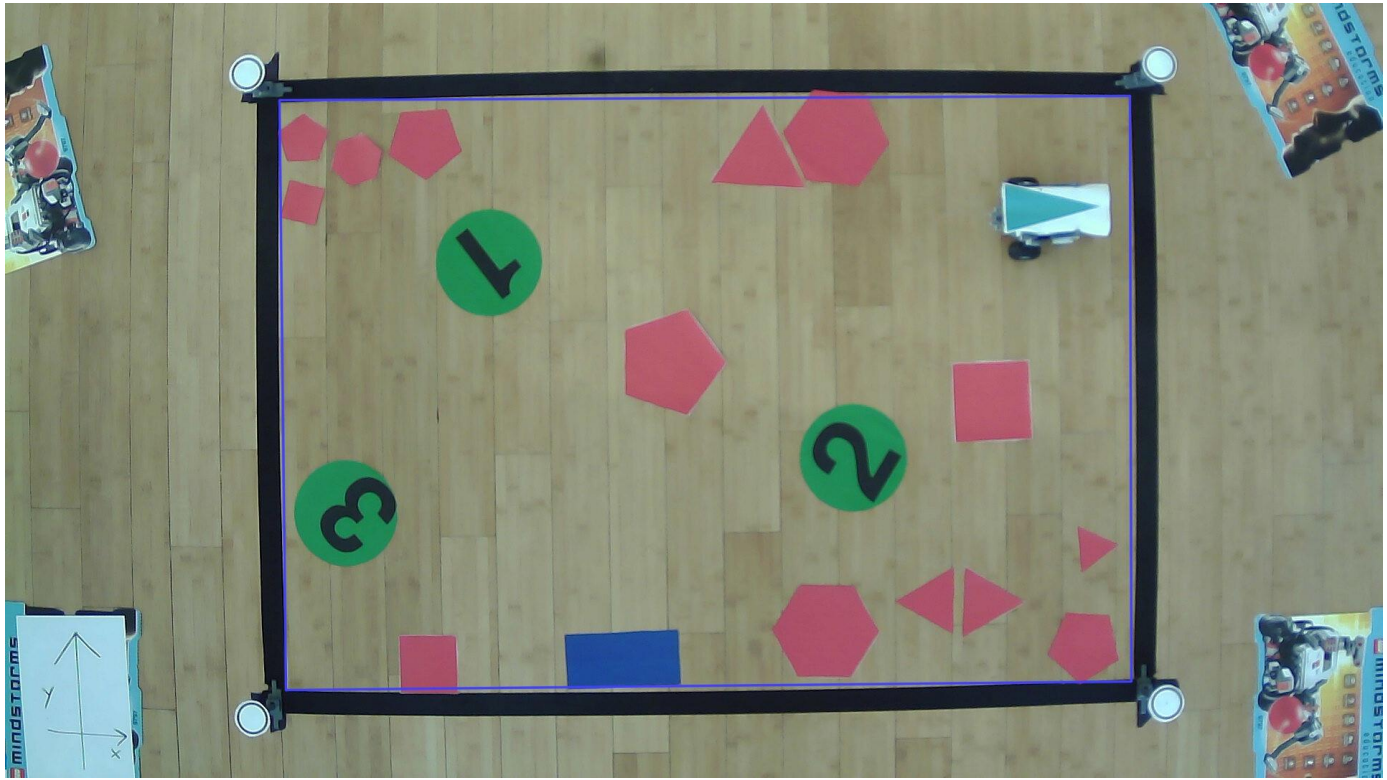
Classes for the image processing



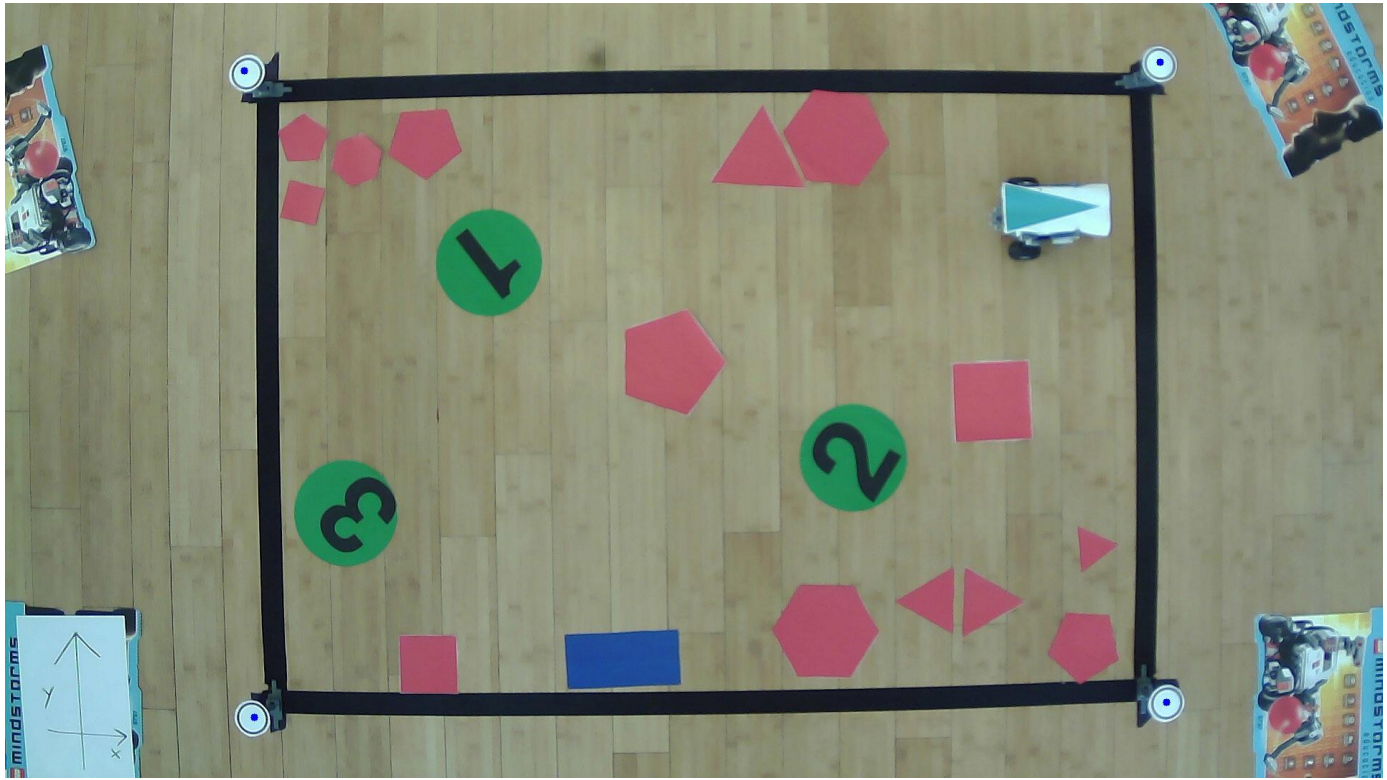
Detector - Board (Adaptive Threshold)



Detector - Board

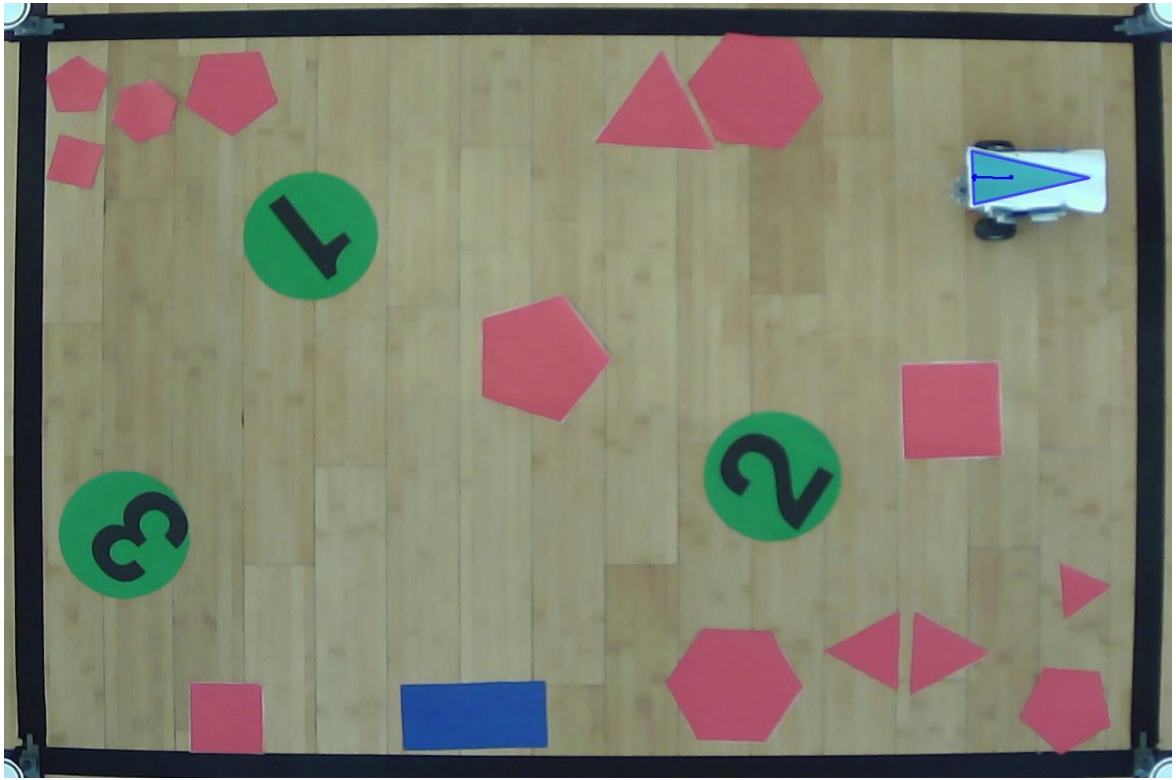


Detector - Robot Plane (Hough Transform)

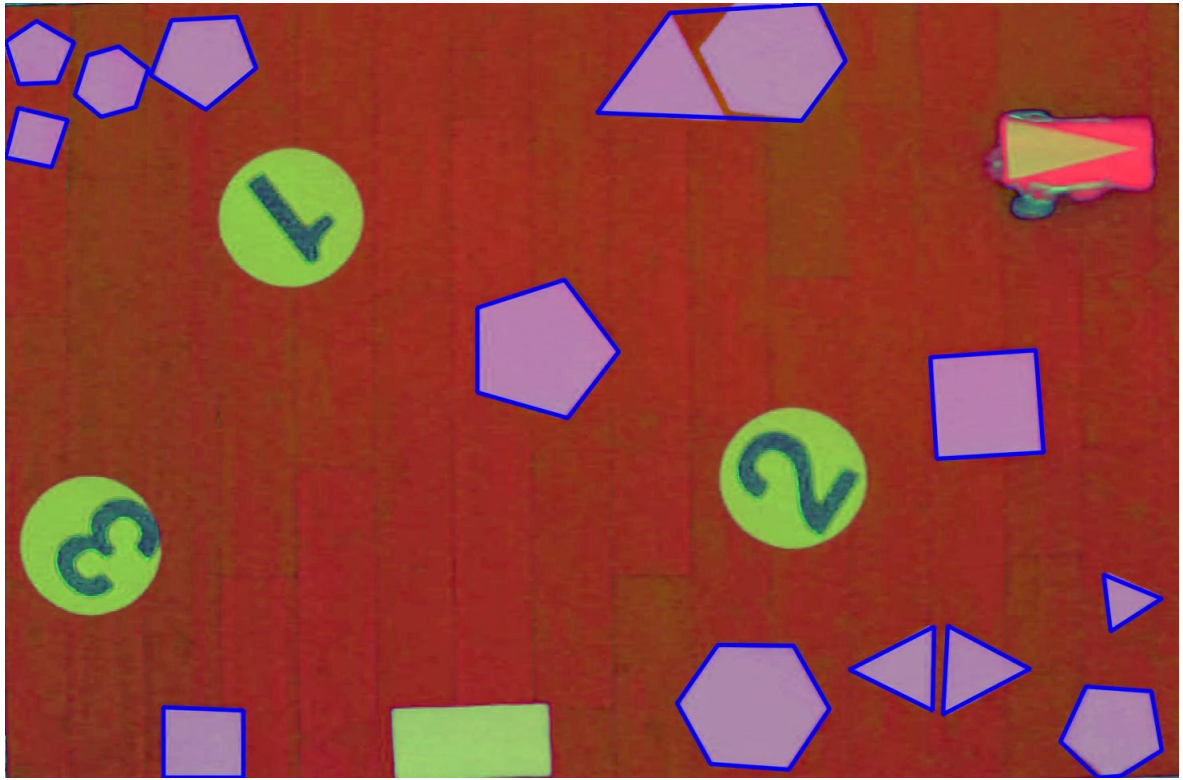


Detector - Localization

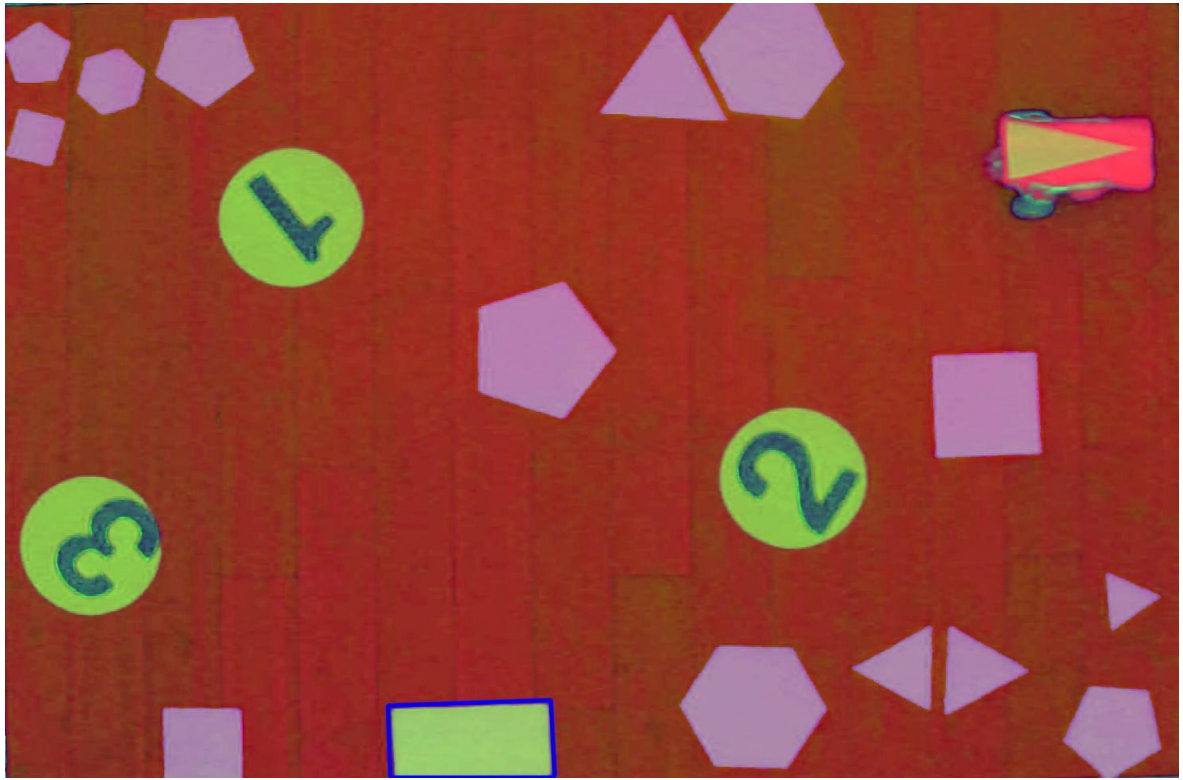
Detector - Localization



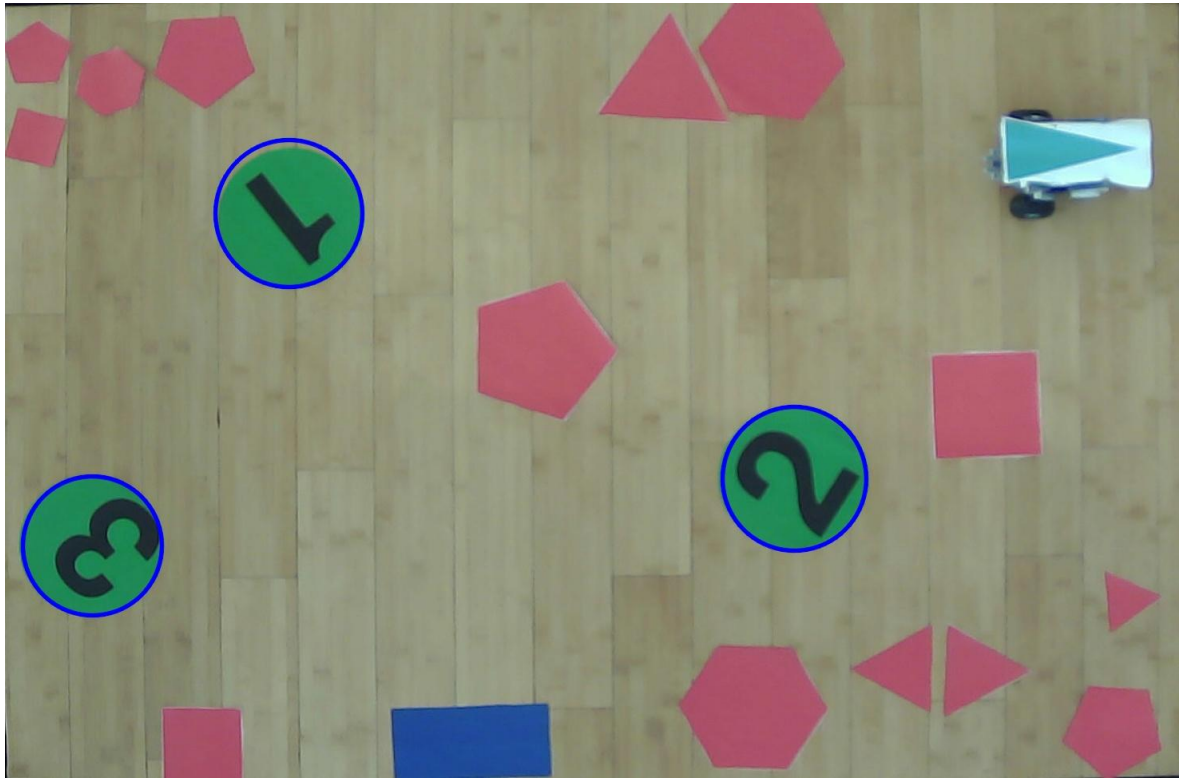
Recognitor - Obstacles (Convex Hull)



Recognitor - Gate



Recognitor - Victims (Hough)



Recognitor - Digits (YUV)

Digit ROI

YUV

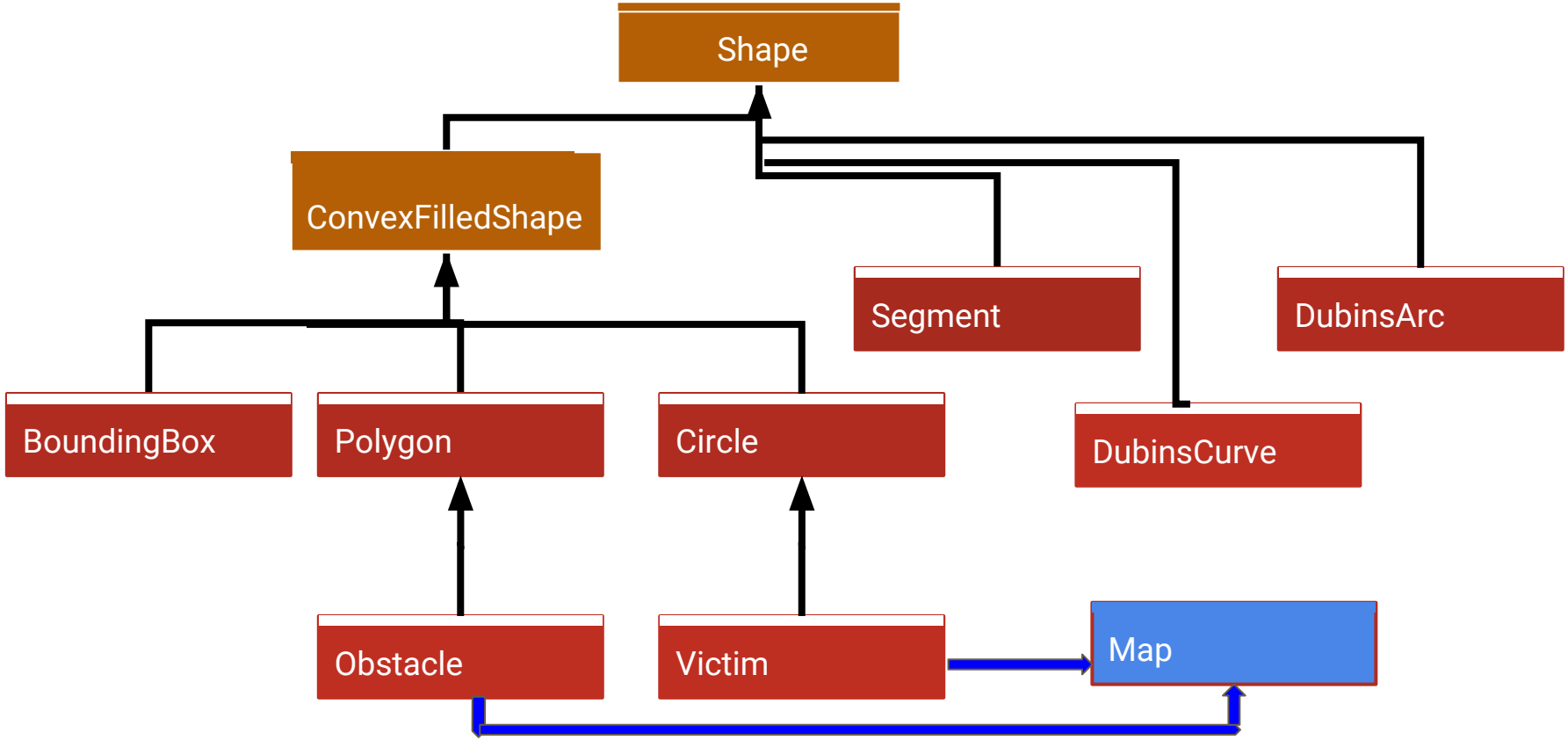
V Channel

Recognitor - Digits (Otsu's thresholding)

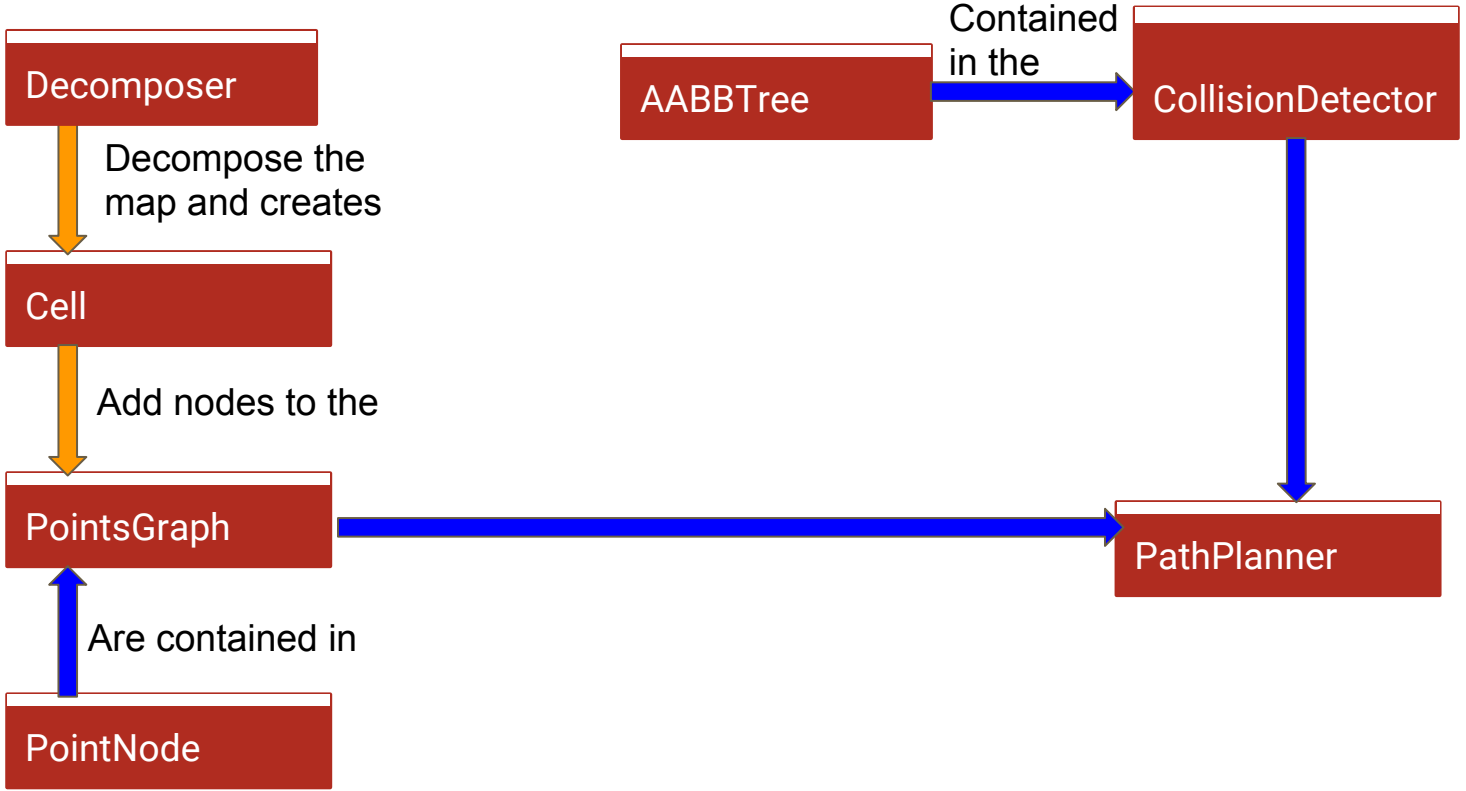
Otsu's Threshold

Centered Digit

Classes for the geometry



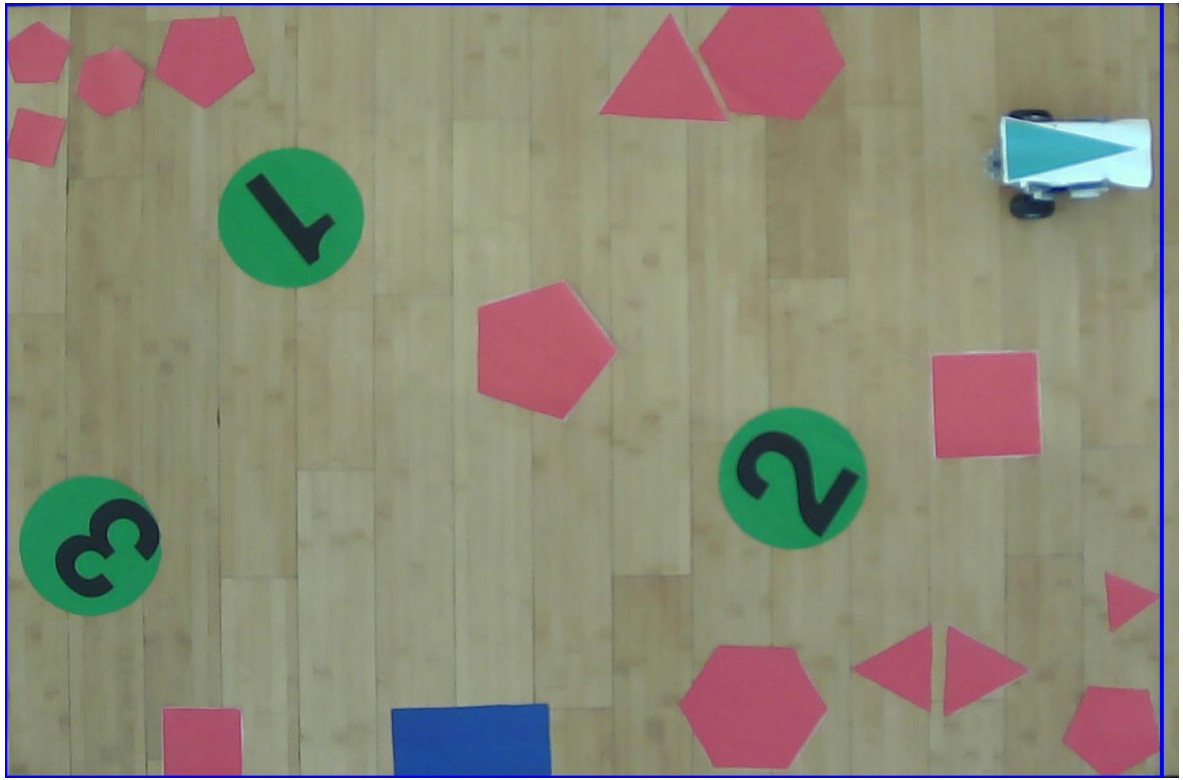
Classes for the path planning



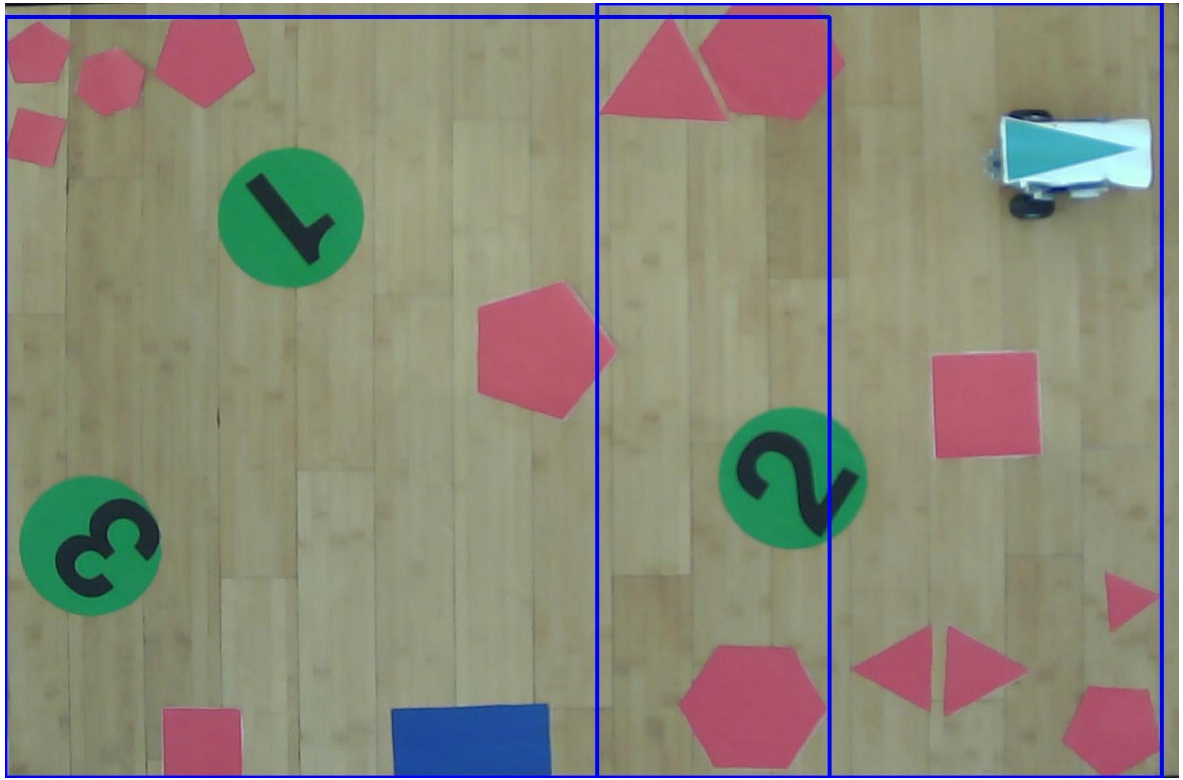
Collisions: AABB

- Binary AABBTree built with Top-Down approach
- Axis choice: x or y axis, chosen to cut the bounding box along its longest dimension
- Sorting strategy: centers of the obstacles sorted by x or y depending on how we’re splitting it
- Splitting strategy: median of the set of obstacles
- Collision check: start from biggest bounding box, test recursively children bounding boxes if it collides

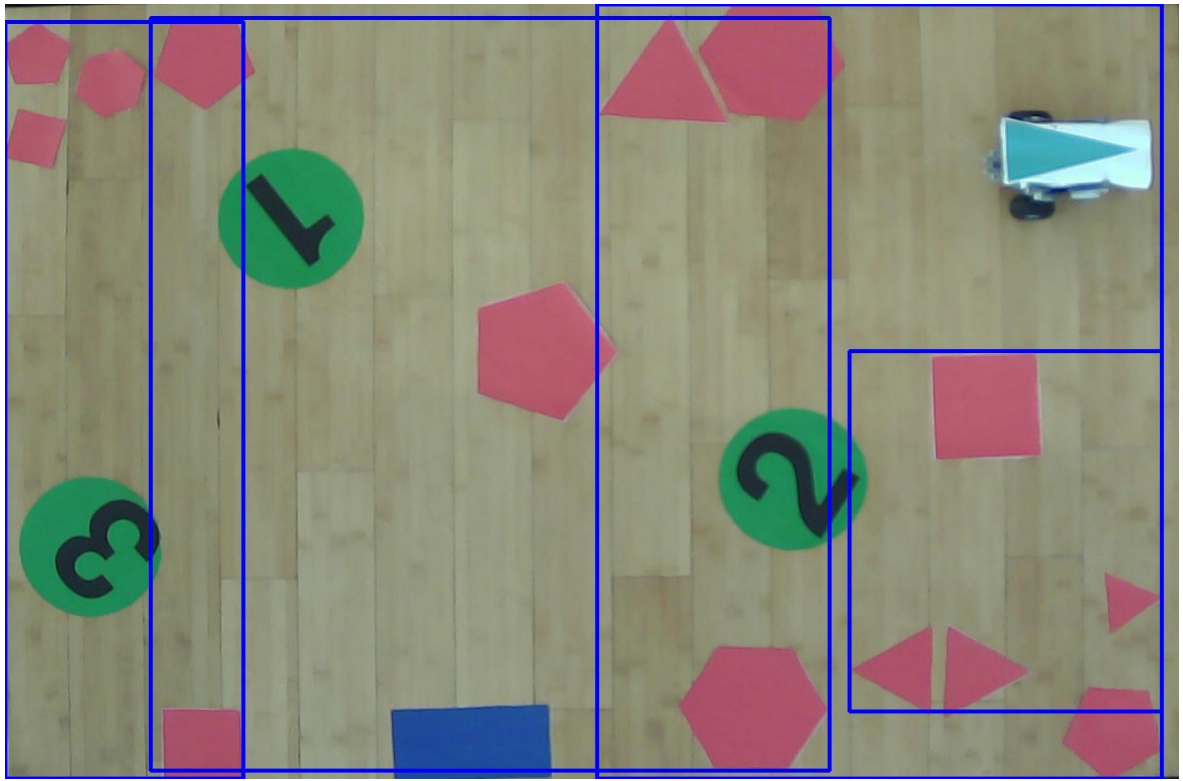
AABB Tree - First BB



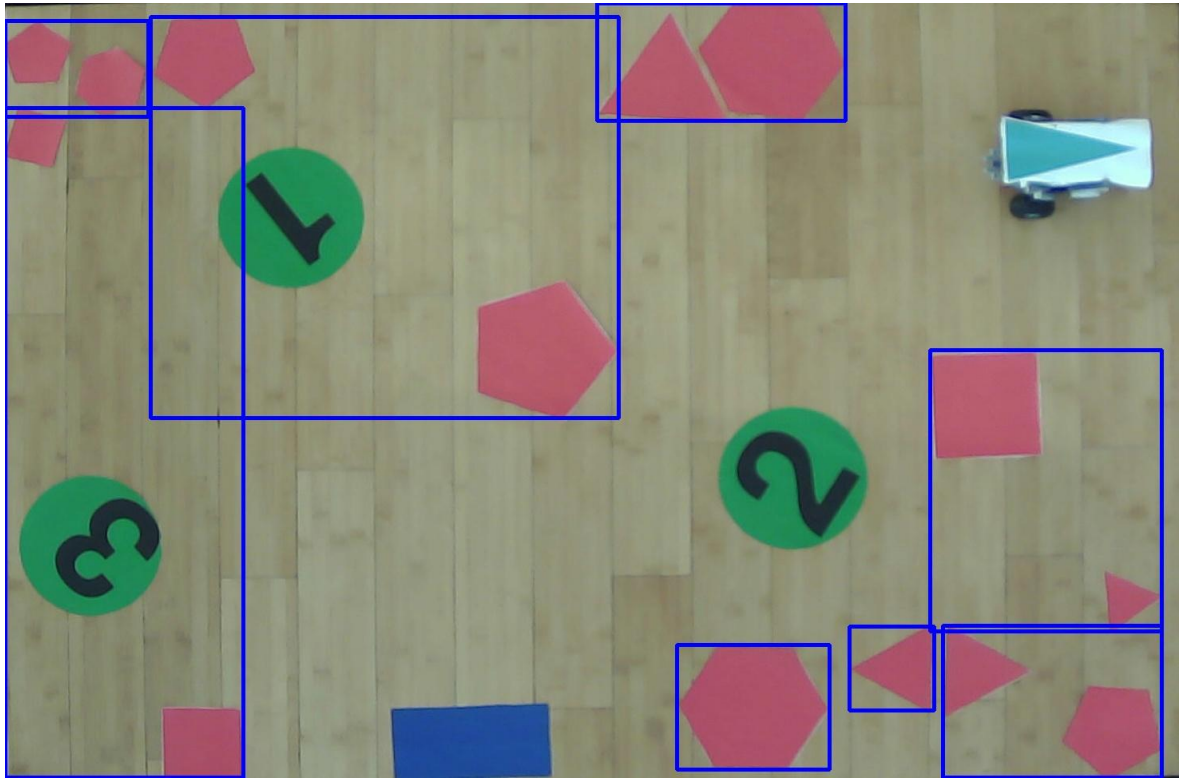
AABB Tree - First Split



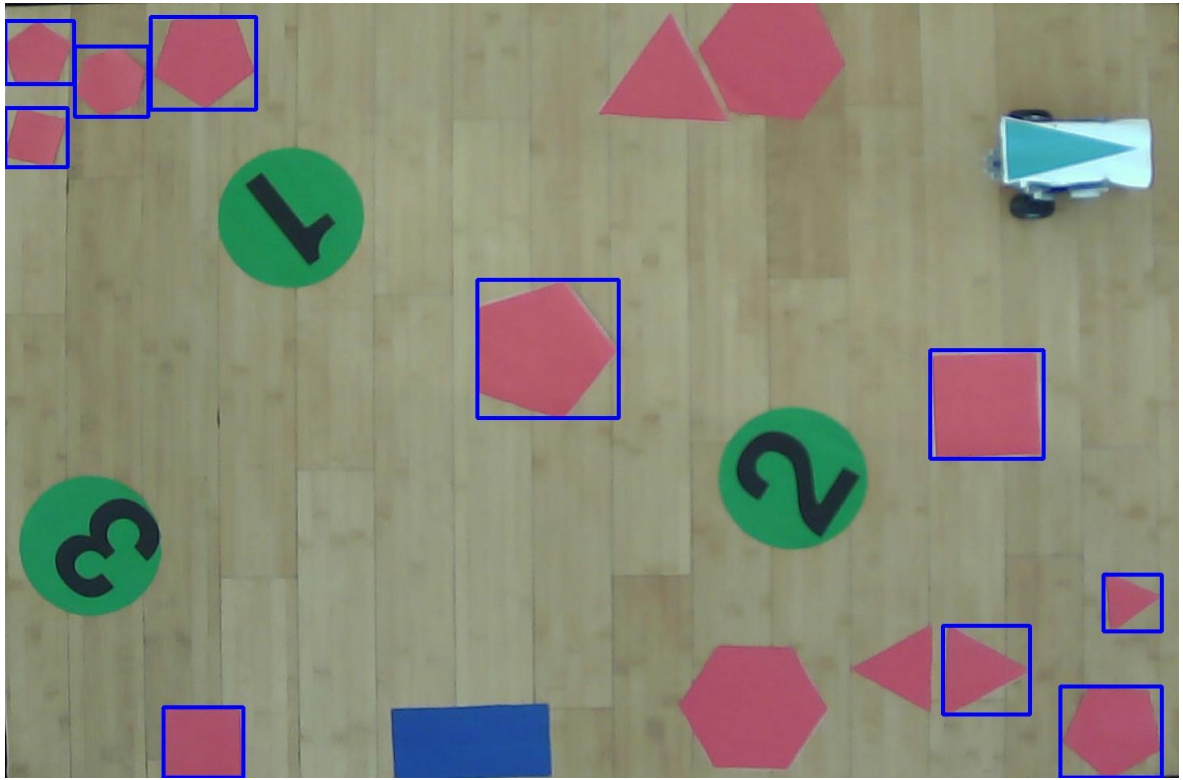
AABB Tree - Second Split



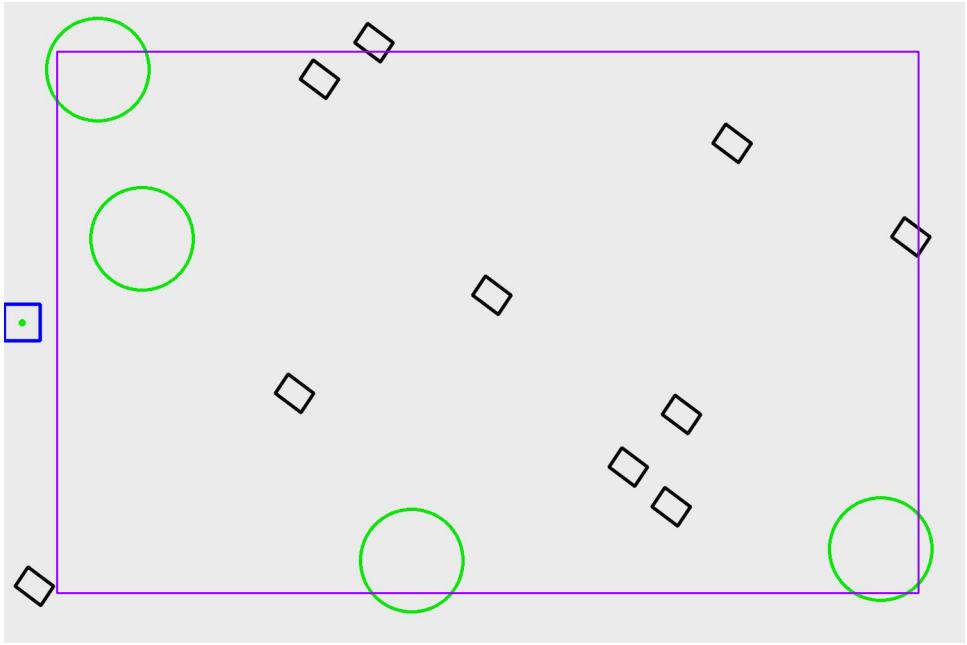
AABB Tree - Third Split

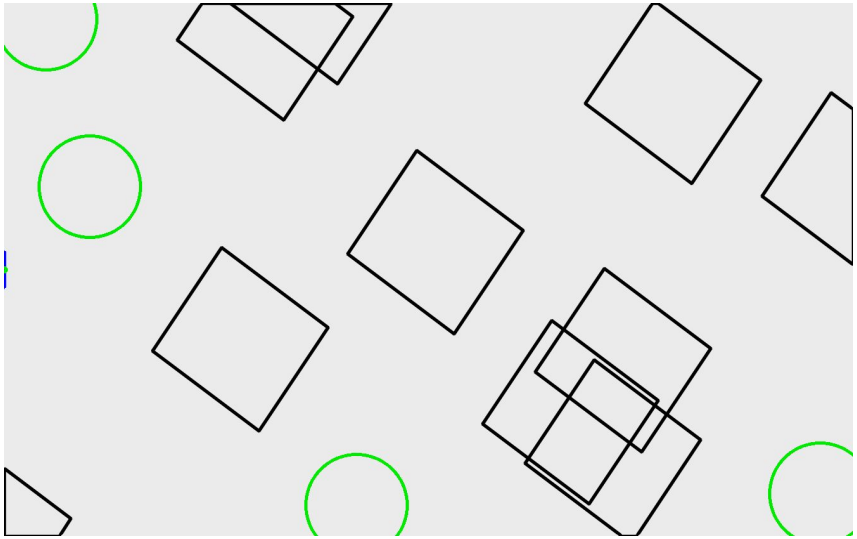


AABB Tree - Last Split

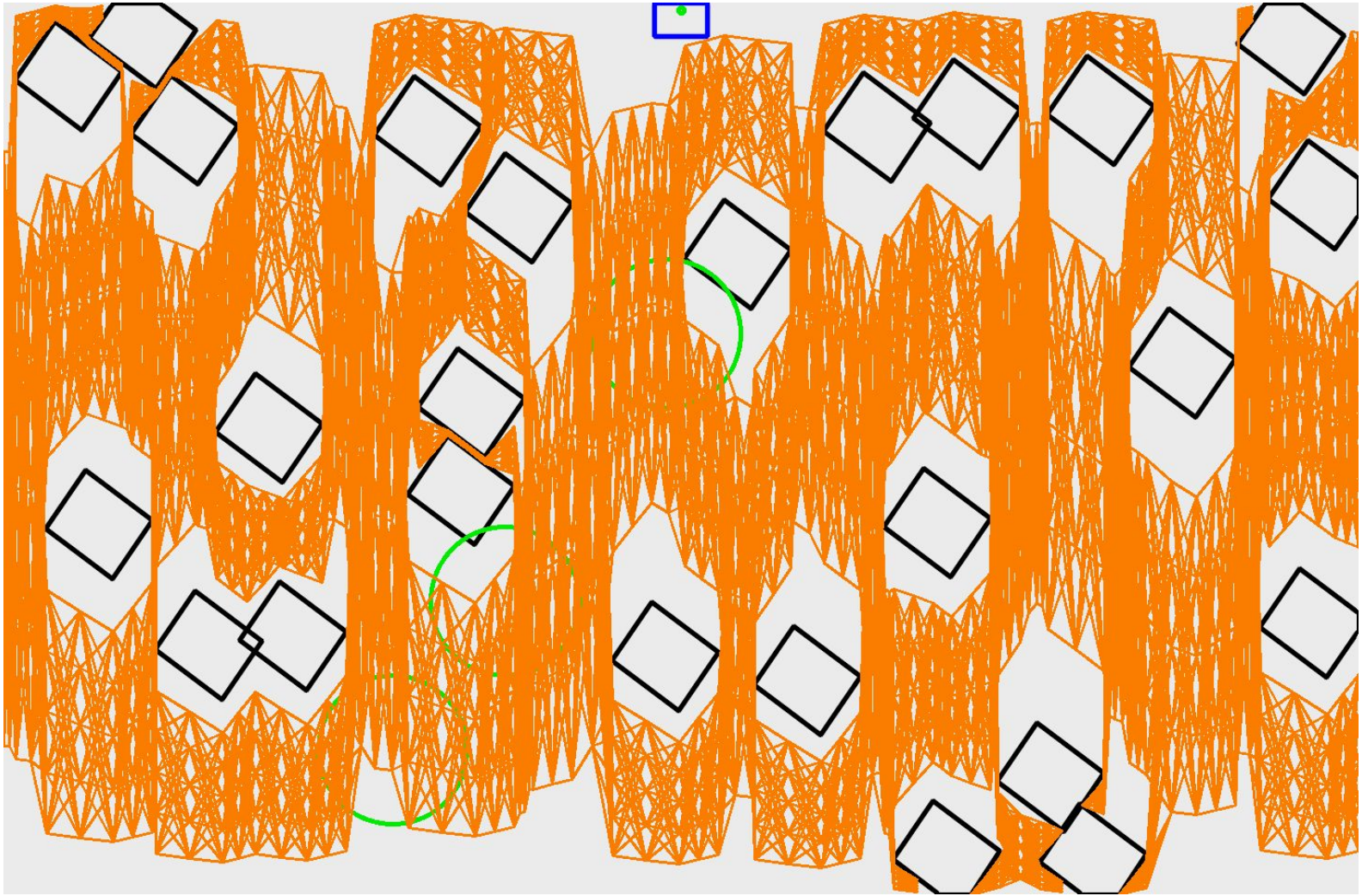


Prepare the map, step 1: clipping





Step 2: LineSweep algorithm



Path Planning

The path planning is divided in these steps:

1. Create a ordered list of points to visit (mission planning)
2. For every point found(path planning):
    a. Compute the best path in the graph from one point to another
    b. Recursively try to go from the initial point to the final with Dubins Curves following the points in the graph

Step 1 : Mission planner

A matrix of the distances(using the graph) is computed:

	Start Point	Victim 1	Victim 2	Victim 3	Goal
Start Point	0	10	5	2	15
Victim 1	10	0	5	7	8
Victim 2	5	5	0	10	10
Victim 3	2	7	10	0	15
Goal	15	8	10	15	0

Mission planner pseudocode

1. Create a copy of the distance table
2. Select the closest victim to the starting point, this will be the first point to visit, eliminate the column of the starting point
3. Eliminate the column of that victim
4. Find the closest victim to this one, that will be the next point to visit
5. If victims are not finished go to step 3
6. Go from the current victim to the goal
7. Update the original distance table eliminating the farthest victim from the starting point and the goal
8. If there is only the goal in the table we finished, otherwise go to step 1

Mission 2 planner pseudocode

```
path = {}
for every victim in the map:
        current position = starting point
        current_path = {}
        for every victim in the map:
                current_path.add(closest victim to current position)
                current_position = closest victim
        current_path.add(goal)
        path.add(current_path)
        remove the victim further away from both the starting point and the goal from the map
find the best path in the path vector based on the travel time minus victim rescued bonus
```

	Start Point	Victim 1	Victim 2	Victim 3	Goal
Start Point	[redacted]	10	5	**2**	15
Victim 1	[redacted]	0	5	7	8
Victim 2	[redacted]	5	0	10	10
Victim 3	[redacted]	7	10	0	15
Goal	[redacted]	8	10	15	0

	Start Point	Victim 1	Victim 2	Victim 3	Goal
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
Victim 1	[redacted]	0	5	[redacted]	8
Victim 2	[redacted]	5	0	[redacted]	10
Victim 3	[redacted]	7	10	[redacted]	15
Goal	[redacted]	8	10	[redacted]	0

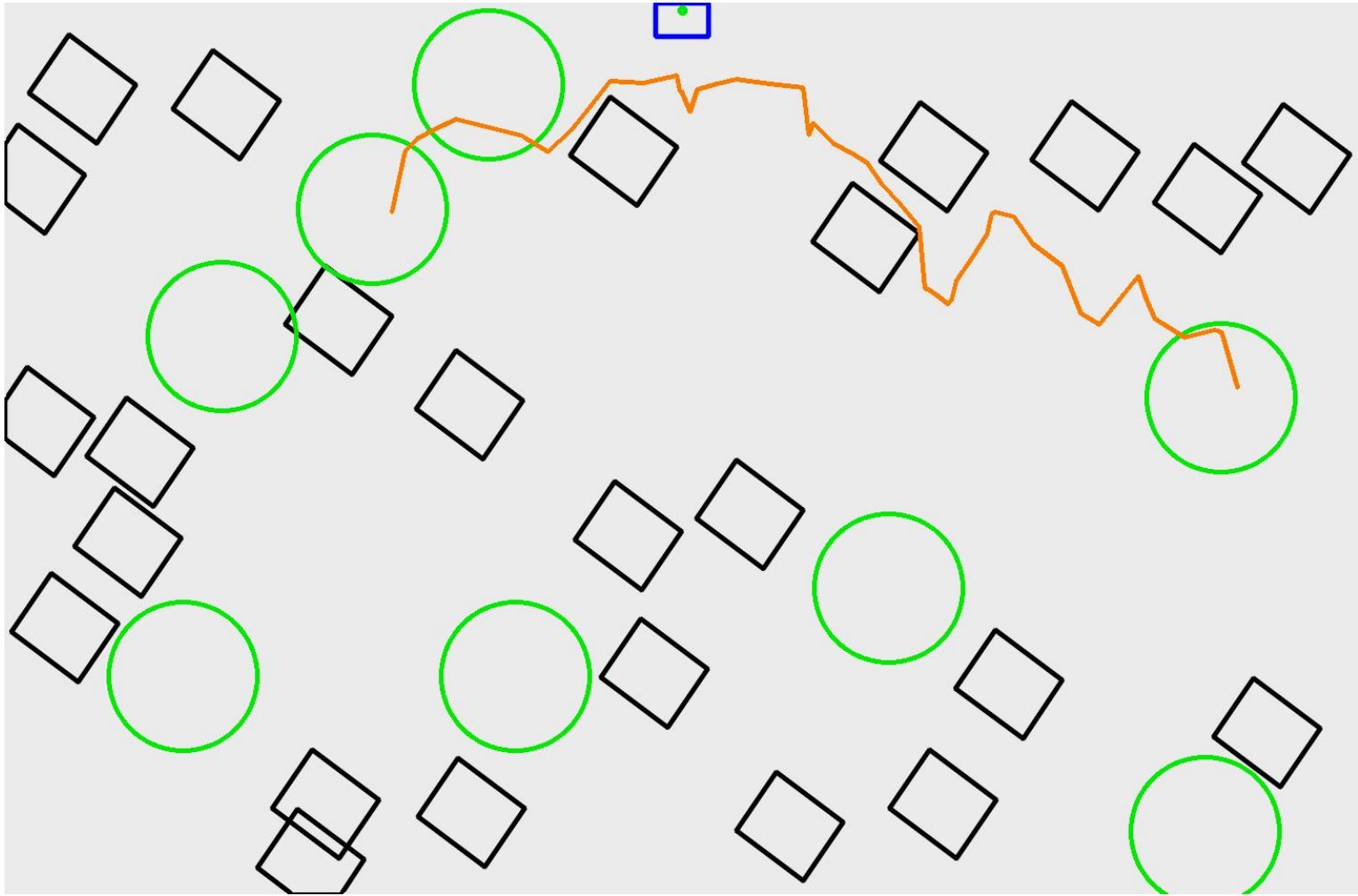
The victim further away from both the goal and the starting point is victim 1 (distance 10 to the starting point and 8 from the goal), so we eliminate it and we repeat the process

New distance table:

	Start Point	Victim 2	Victim 3	Goal
Start Point	0	5	2	15
Victim 2	5	0	10	10
Victim 3	2	10	0	15
Goal	15	10	15	0

Step 2a

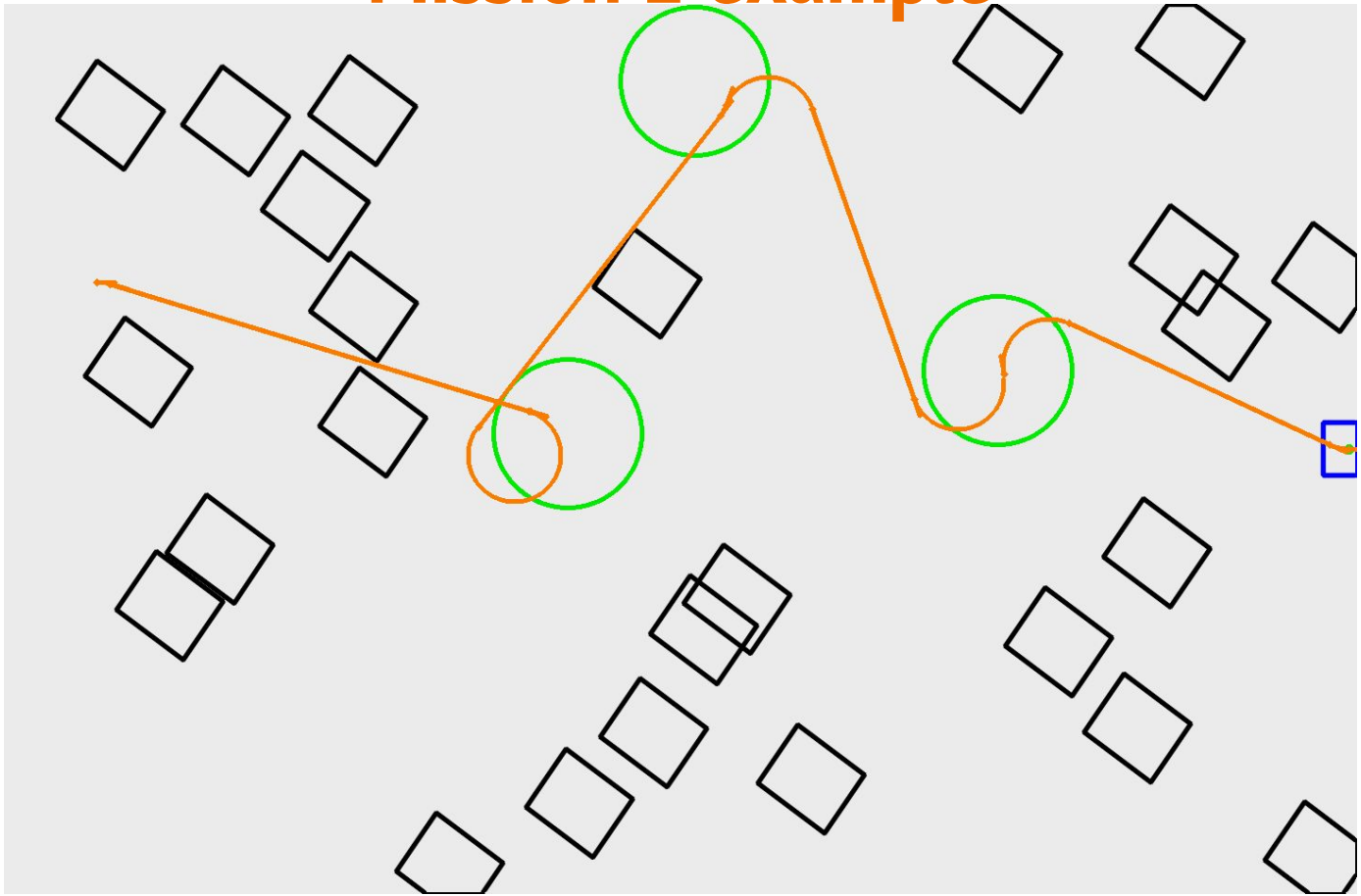
Graph path:

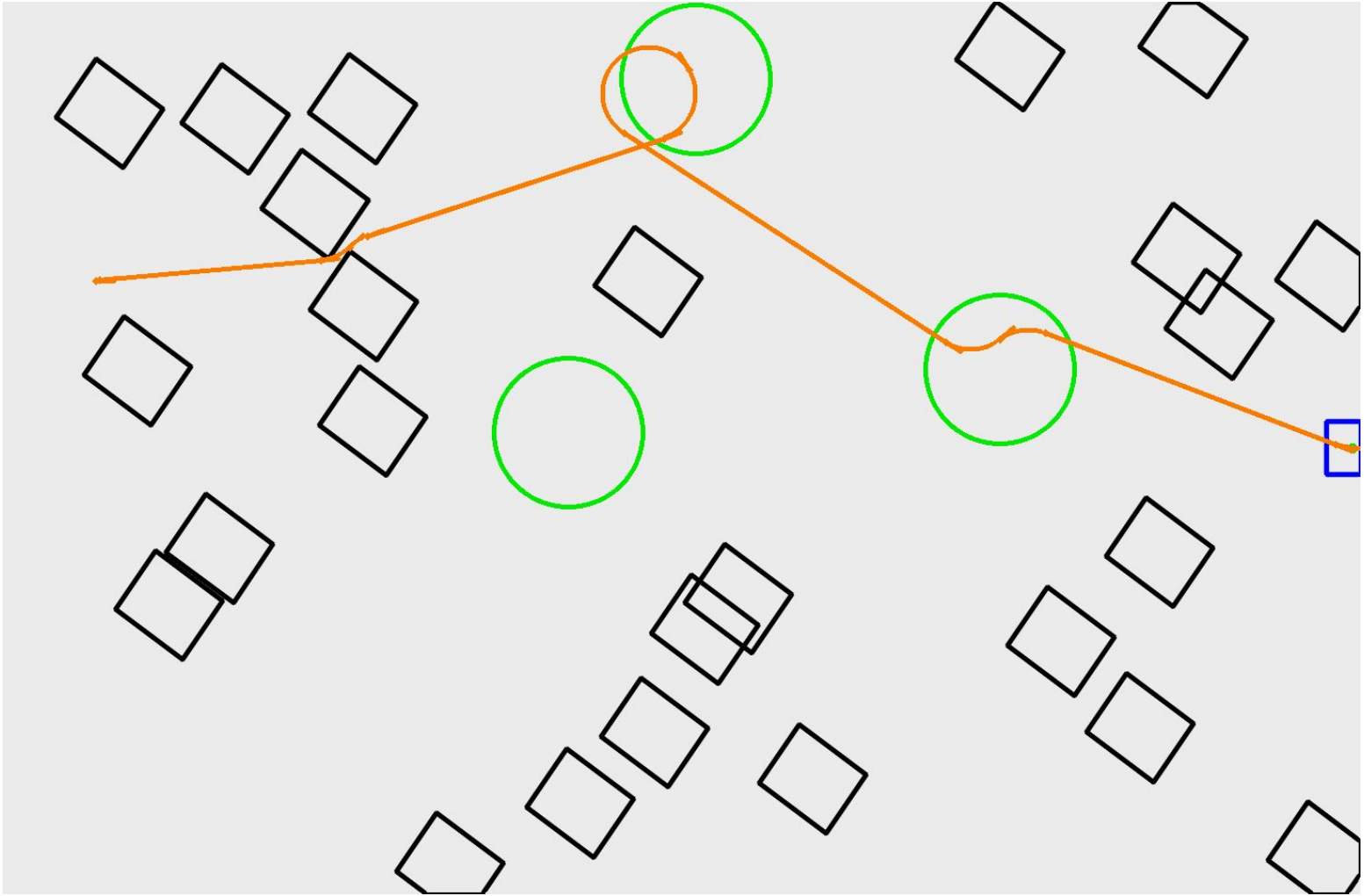


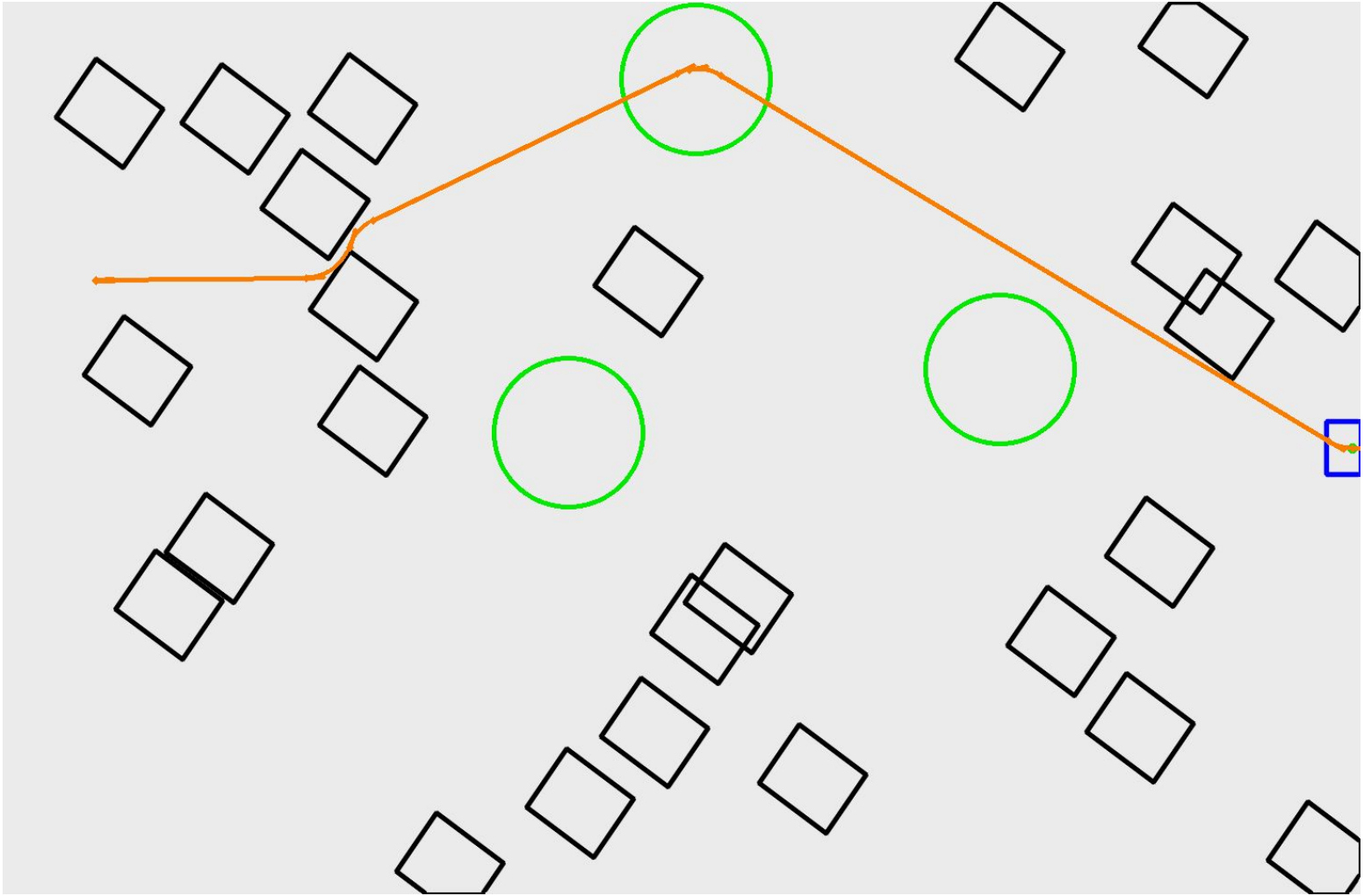
Step 2b: recursive part

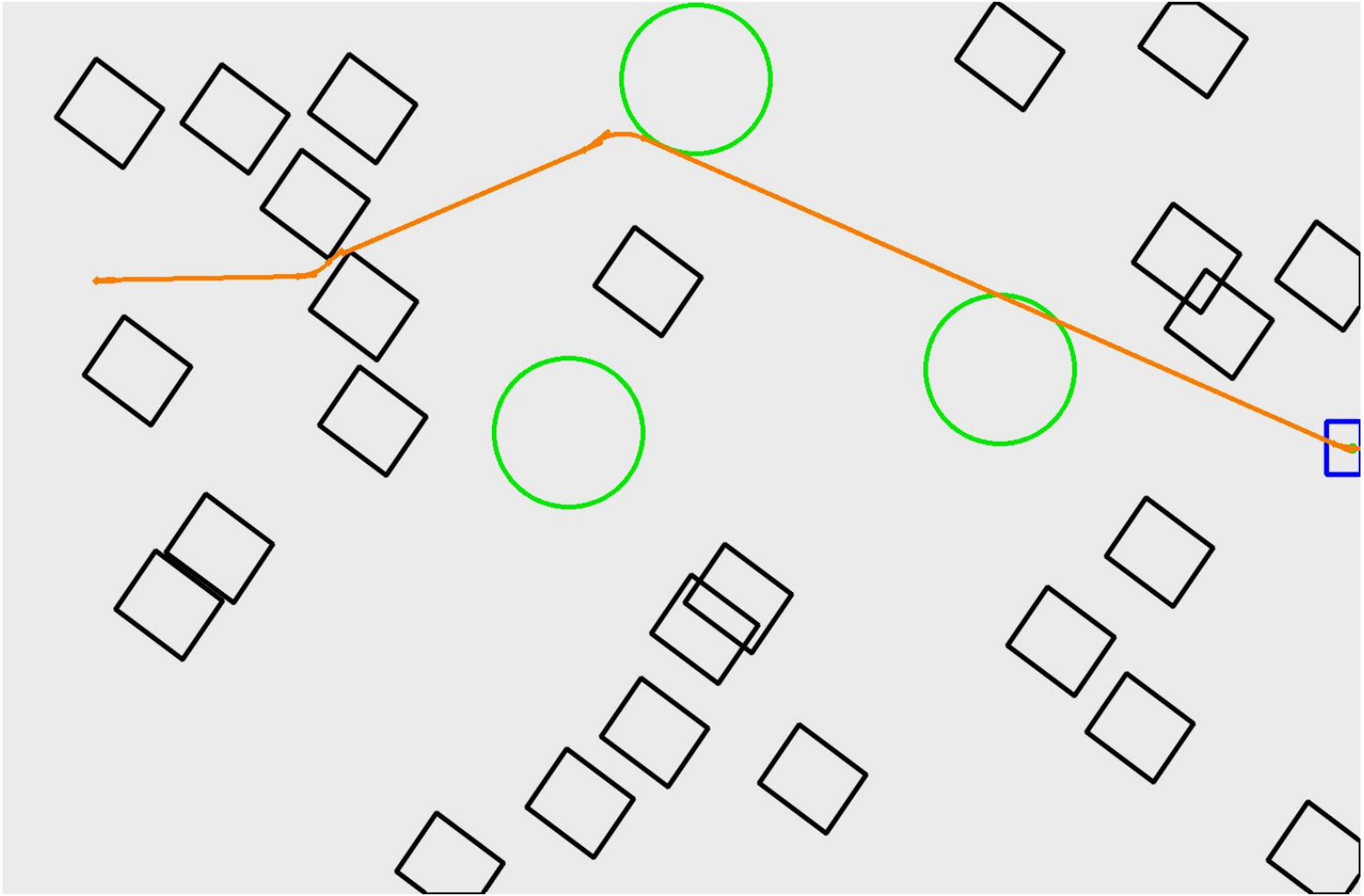
```
bool recursiveFunction(start_index,start_angle,end_index,end_angle,path,*solution):
    for(i = 0; i < N; i++) {
        end_angle = randomAngle();
        DubinsCurve dest_curve* = NULL;
        success = collisionDetector.getCurveWithoutCollision(path[start_index], start_angle,
            path[end_index],end_angle,dest_curve);
        if(success) { solution->append(*dest_curve); return true;}
    }
    if(end_index - start_index <= 1) { return false;}
    else {
        int middlepoint = (start_index+end_index)/2;
        if(recursiveFunction(start_index, start_angle, middlepoint,NULL,solution) &&
        recursiveFunction(middlepoint, solution.getLastAngle(),end_index,end_angle,solution) {
            return true;
        }else {  return false;  }
    }
```

Mission 2 example

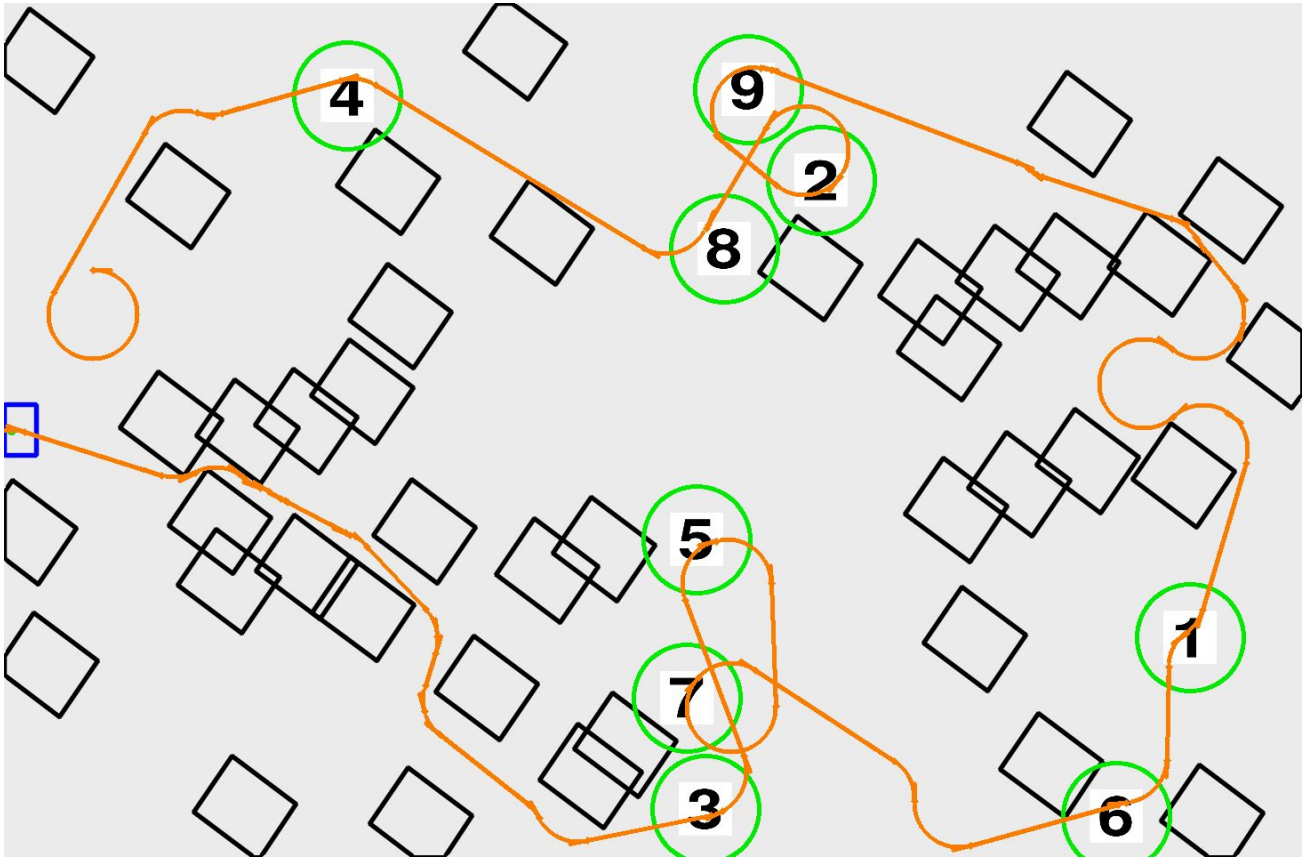


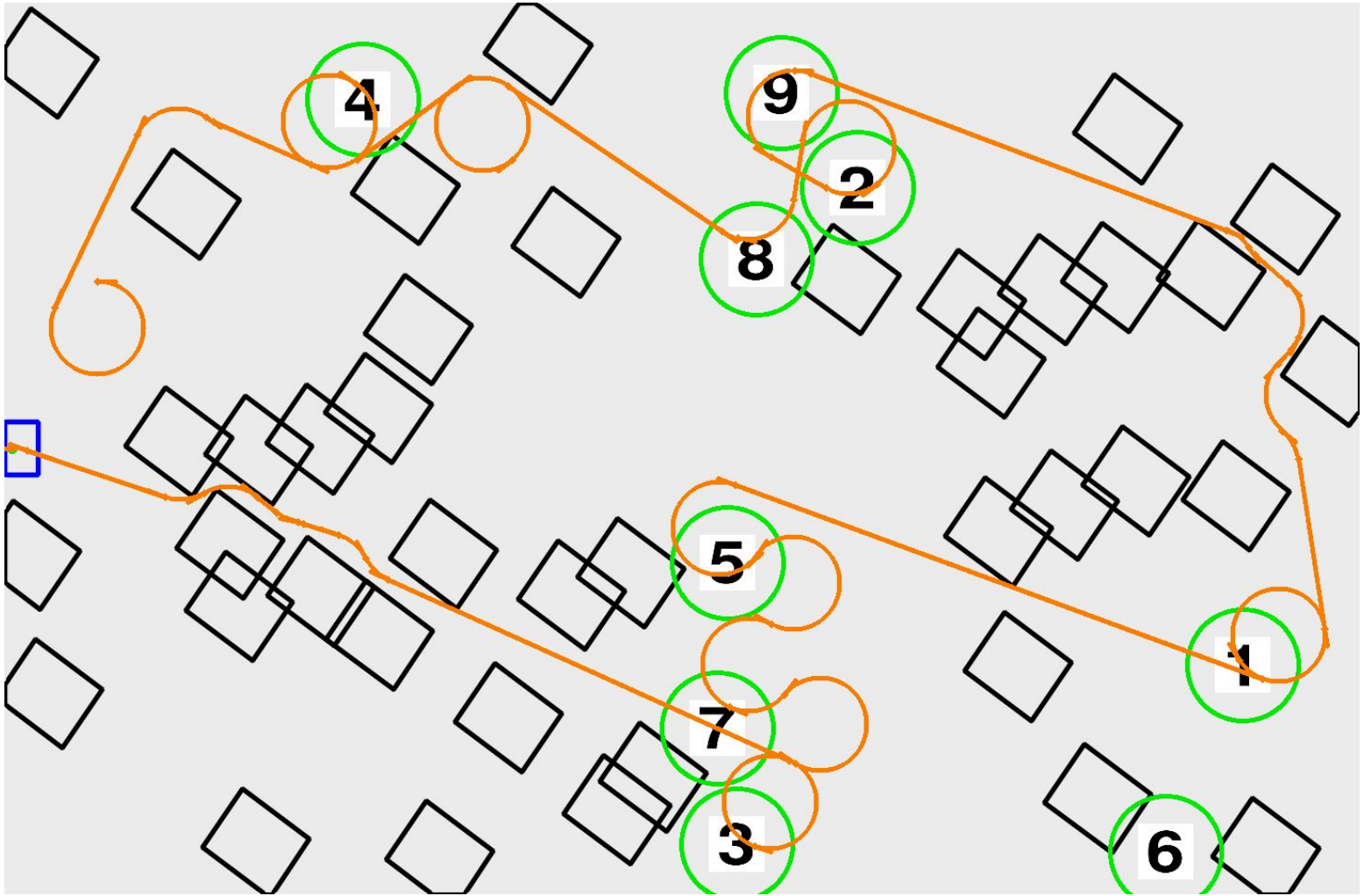


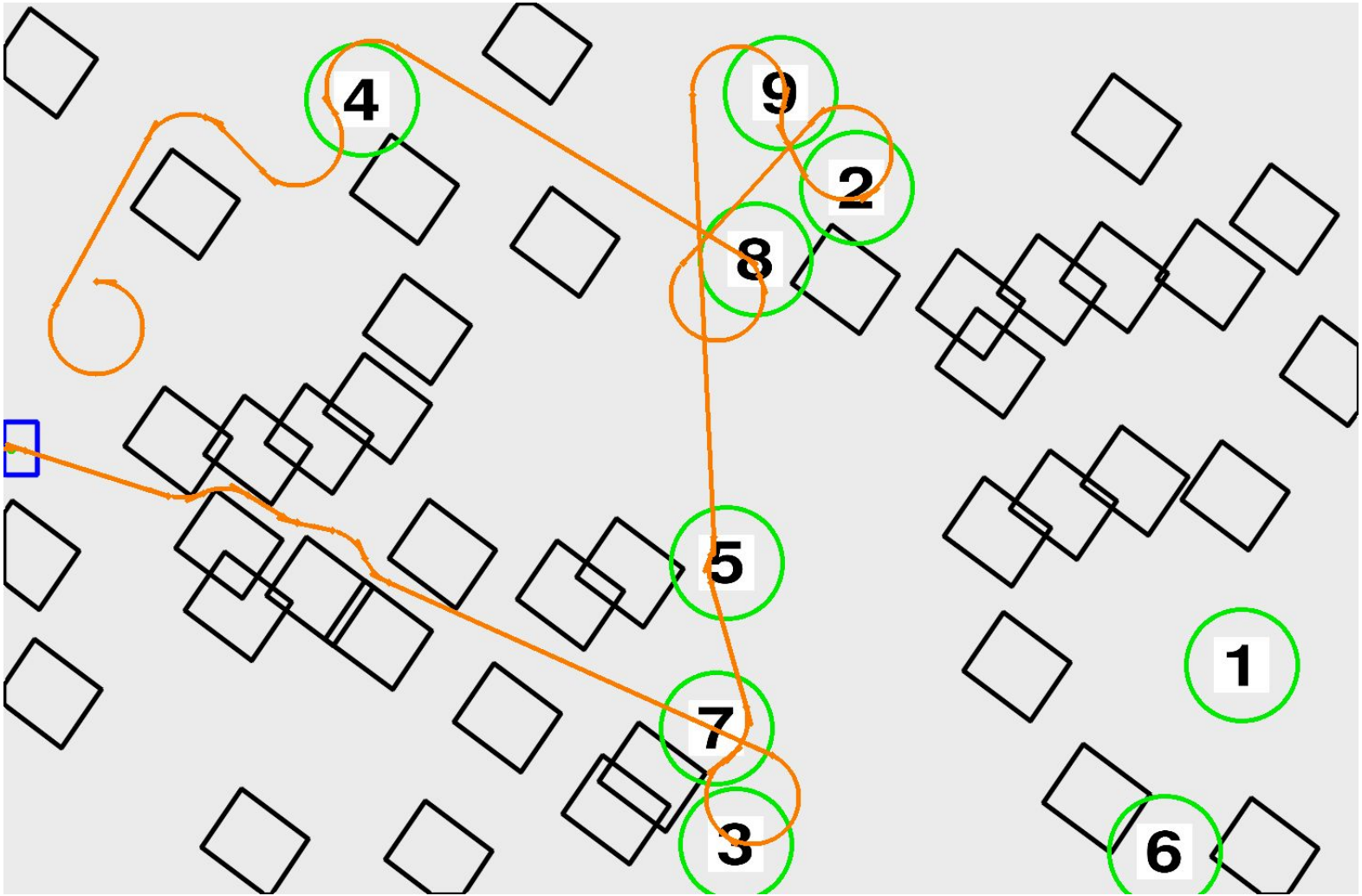




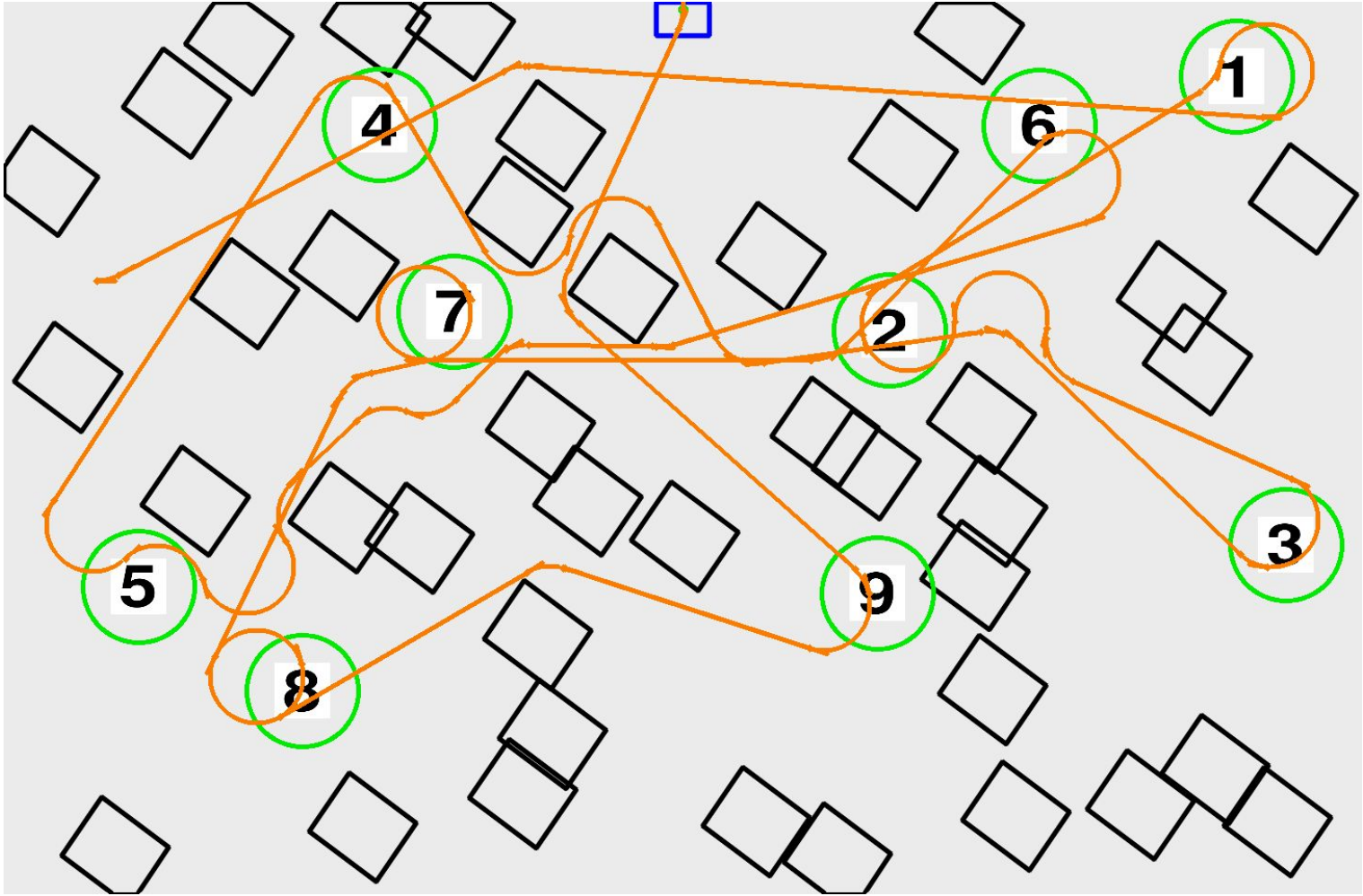
Another mission 2 example

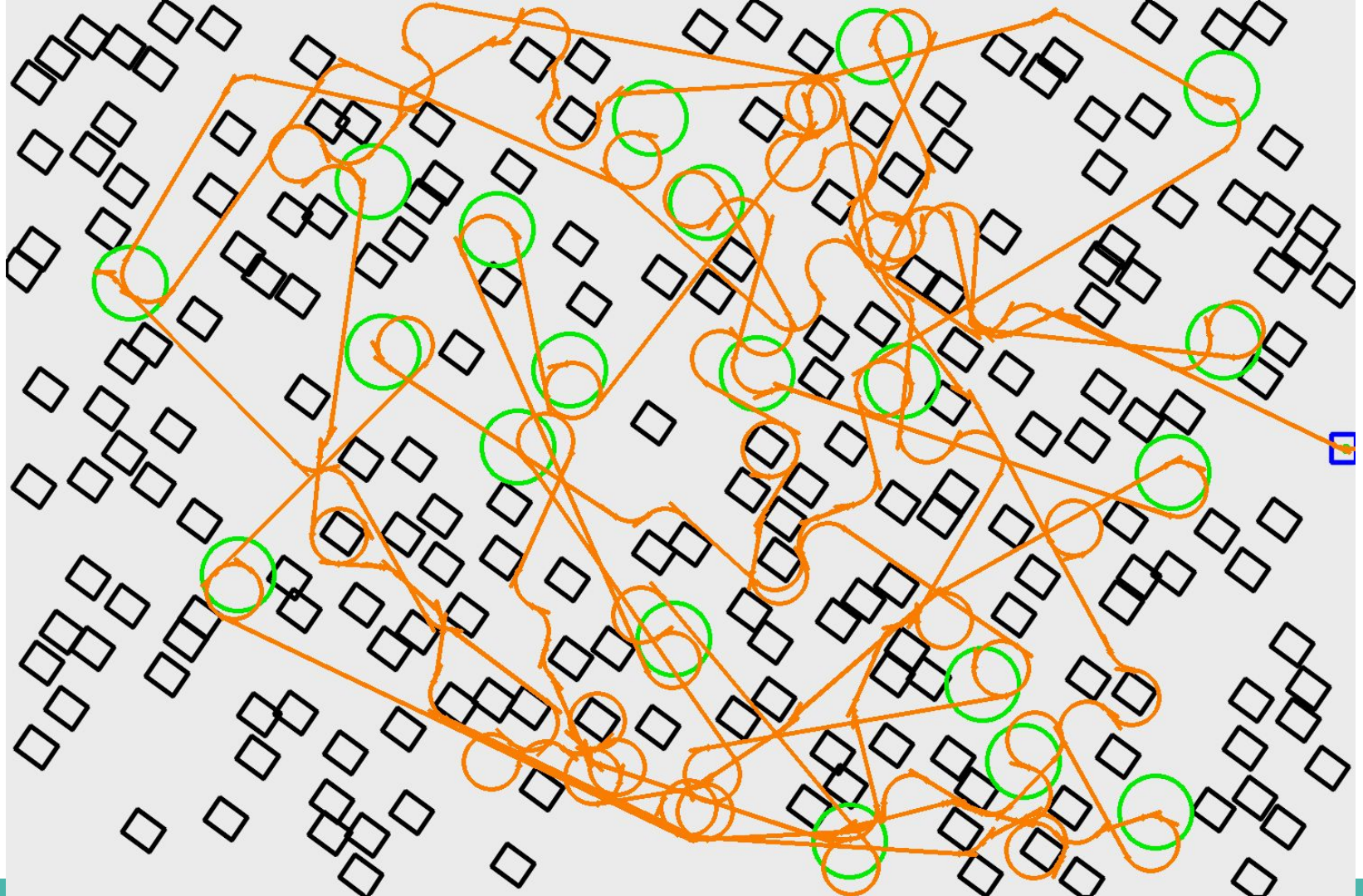



4
9
2
8
5
1
7
3
6


4
9
2
8
5
1
7
3
6

Mission 1 example





Laboratory of Applied Robotic Final Presentation

Sergio Catalano,
Simone Zamboni
25/01/2019