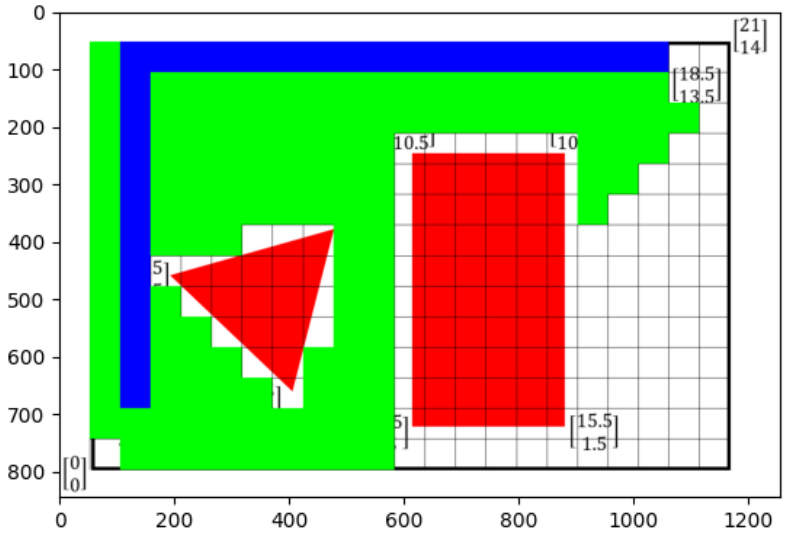
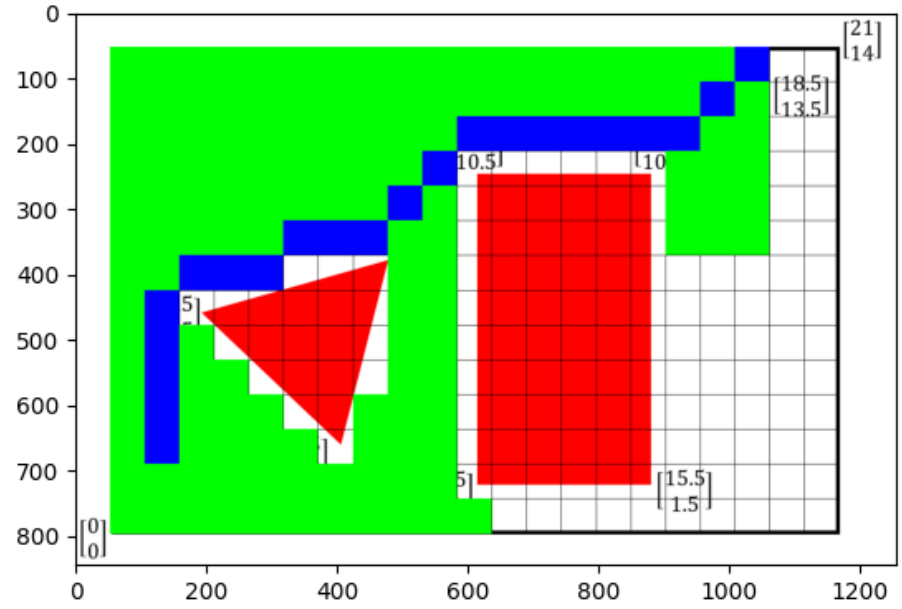
TEL280 Assignment 5

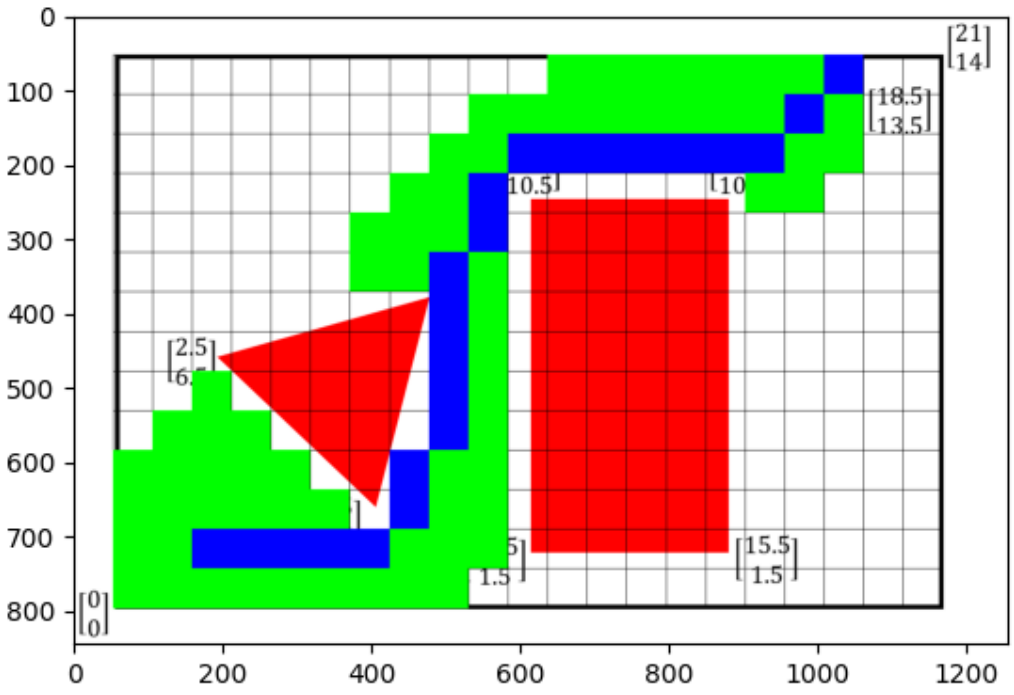
A* path planer with the Manhattan distance heuristic, moving the robot just up, down, left, right. Green is cells that are explored and blue is the backtracking path:




A* path planer with the Manhattan distance heuristic, moving the robot in the eight neighboring cell. Green is cells that are explored and blue is the backtracking path:




A* path planer with the Diagonal distance heuristic, moving the robot in the eight neighboring cell. Green is cells that are explored and blue is the backtracking path:




Code used to calculate A* with given image.

```
import skimage
import matplotlib.pyplot as plt
import numpy as np


class Squared(object):
    def __init__(self, x, y, x_start, x_stop, y_start, y_stop):
        self.x = x
        self.y = y
        self.x_start = x_start
        self.x_stop = x_stop
        self.y_start = y_start
        self.y_stop = y_stop
        self.free = True
        self.goal = False
        self.start = False
        self.G = 0
        self.H = 0
        self.F = self.G + self.H
        self.parent = None

    def check_box(self, image):
        for k in np.arange(self.x_start, self.x_stop, 1):
            for o in np.arange(self.y_start, self.y_stop, 1):
                red = image[int(o), int(k), 0]
                green = image[int(o), int(k), 1]
```

```
                blue = image[int(o), int(k), 2]
                if red > 200 and green < 1 and blue < 1:
                    self.free = False
                    break

                elif green > 200 and red < 200 and blue < 200:
                    self.goal = True
                    break

                elif blue > 200 and red < 200 and green < 200:
                    self.start = True
                    break


if __name__ == '__main__':

    # %% Initialising

    im = skimage.io.imread('Assignment5_tel280.png')
    xstart = x__start = 0
    xstop = x__stop = 1169
    ystart = 0
    ystop = y__stop = 797
    grid_sizex = 0
    stepx = (xstop-xstart)/22
    grid_sizey = 0
    stepy = (ystop-ystart)/15

    list_of_nodes = []
    x__stop = x__start + stepx
    for i in range(21):
        x__start += stepx
        x__stop += stepx
        y__start = ystart
        y__stop = y__start + stepy
        for j in range(14):
            y__start += stepy
            y__stop += stepy
            list_of_nodes.append(Squared(i, j, x__start, x__stop, y__start,
y__stop))
            im[int(y__start):int(y__stop), int(x__start)] = 0
            im[int(y__start):int(y__stop), int(x__stop)] = 0
            im[int(y__start), int(x__start):int(x__stop)] = 0
            im[int(y__start), int(x__stop):int(x__stop)] = 0

    for element, node in enumerate(list_of_nodes):
        node.check_box(im)

    # %% A* mapping
    plt.imshow(im)
    plt.show()
    im2 = im

    open_list = []
    closed_list = []
    for i in range(len(list_of_nodes)):
        if list_of_nodes[i].start:
            start_square = list_of_nodes[i]
            open_list.append(list_of_nodes[i])

    for i in range(len(list_of_nodes)):
```

```
        if list_of_nodes[i].goal:
            goal_square = list_of_nodes[i]

    run = True
    while run:
        open_list.sort(key=lambda x: x.F)
        q = open_list[0]

        # find neighbour_cell
        neighbour_cell = []
        for i in range(len(list_of_nodes)):
            if list_of_nodes[i].y == q.y - 1 \
                    and list_of_nodes[i].x == q.x \
                    and list_of_nodes[i].free:
                upper = list_of_nodes[i]

            elif list_of_nodes[i].y == q.y - 1 \
                    and list_of_nodes[i].x == q.x + 1 \
                    and list_of_nodes[i].free:
                upper_right = list_of_nodes[i]

            elif list_of_nodes[i].x == q.x + 1 \
                    and list_of_nodes[i].y == q.y \
                    and list_of_nodes[i].free:
                right = list_of_nodes[i]

            elif list_of_nodes[i].x == q.x + 1 \
                    and list_of_nodes[i].y == q.y + 1 \
                    and list_of_nodes[i].free:
                lower_right = list_of_nodes[i]

            elif list_of_nodes[i].y == q.y + 1 \
                    and list_of_nodes[i].x == q.x \
                    and list_of_nodes[i].free:
                lower = list_of_nodes[i]

            elif list_of_nodes[i].y == q.y + 1 \
                    and list_of_nodes[i].x == q.x - 1 \
                    and list_of_nodes[i].free:
                lower_left = list_of_nodes[i]

            elif list_of_nodes[i].x == q.x - 1 \
                    and list_of_nodes[i].y == q.y \
                    and list_of_nodes[i].free:
                left = list_of_nodes[i]

            elif list_of_nodes[i].x == q.x - 1 \
                    and list_of_nodes[i].y == q.y - 1 \
                    and list_of_nodes[i].free:
                upper_left = list_of_nodes[i]

        neighbour_cell = [upper, upper_left, left, lower_left,
                          lower, lower_right, right, upper_right]

        for i in range(len(neighbour_cell)):

            dx = abs(goal_square.x - neighbour_cell[i].x)
            dy = abs(goal_square.y - neighbour_cell[i].y)
            neighbour_cell[i].H = (dx + dy)  # + (np.sqrt(2) - 2) * min(dx,
dy)  # Only used for Diagonal distance
```

```
            dx = abs(start_square.x - neighbour_cell[i].x)
            dy = abs(start_square.y - neighbour_cell[i].y)
            neighbour_cell[i].G = (dx + dy)  # + (np.sqrt(2) - 2) * min(dx,
dy)  # Only used for Diagonal distance

            neighbour_cell[i].F = neighbour_cell[i].G + neighbour_cell[i].H


im[int(neighbour_cell[i].y_start):int(neighbour_cell[i].y_stop),
            int(neighbour_cell[i].x_start):int(neighbour_cell[i].x_stop),
2] = 0

im[int(neighbour_cell[i].y_start):int(neighbour_cell[i].y_stop),
            int(neighbour_cell[i].x_start):int(neighbour_cell[i].x_stop),
0] = 0

im[int(neighbour_cell[i].y_start):int(neighbour_cell[i].y_stop),
            int(neighbour_cell[i].x_start):int(neighbour_cell[i].x_stop),
1] = 255

            if neighbour_cell[i].parent is None:
                neighbour_cell[i].parent = q

        temp = []
        not_add = []
        for i in range(len(neighbour_cell)):
            for j in range(len(open_list)):
                """if a node with the same position as successor is in the
OPEN list which
                has a lower f than successor, skip this successor"""
                if (neighbour_cell[i].y == open_list[j].y
                        and neighbour_cell[i].x == open_list[j].x
                        and neighbour_cell[i].F >= open_list[j].F):
                    not_add.append(neighbour_cell[i])

                elif (neighbour_cell[i].y == open_list[j].y
                      and neighbour_cell[i].x == open_list[j].x
                      and neighbour_cell[i].F < open_list[j].F):
                    open_list.remove(open_list[j])

            for j in range(len(closed_list)):
                """if a node with the same position as successor  is in the
CLOSED list which
                has a lower f than successor, skip this successor"""
                if (neighbour_cell[i].y == closed_list[j].y
                        and neighbour_cell[i].x == closed_list[j].x
                        and neighbour_cell[i].F >= closed_list[j].F):
                    not_add.append(neighbour_cell[i])

                elif (neighbour_cell[i].y == closed_list[j].y
                      and neighbour_cell[i].x == closed_list[j].x
                      and neighbour_cell[i].F < closed_list[j].F):
                    open_list.remove(closed_list[j])

            temp.append(neighbour_cell[i])

        for i in range(len(not_add)):
            temp.remove(not_add[i])

        for i in range(len(temp)):
            open_list.append(temp[i])
```

```
        open_list.remove(q)
        closed_list.append(q)

        for i in range(len(neighbour_cell)):
            if neighbour_cell[i].goal:
                closed_list.append(neighbour_cell[i])
                run = False

    plt.imshow(im)
    plt.show()

    def callback(cell, start_squar, parentlist, im2):
        if start_squar.x == cell.x and start_squar.y == cell.y:
            return parentlist
        parent = cell.parent
        im2[int(cell.y_start):int(cell.y_stop),
int(cell.x_start):int(cell.x_stop), 2] = 255
        im2[int(cell.y_start):int(cell.y_stop),
int(cell.x_start):int(cell.x_stop), 0] = 0
        im2[int(cell.y_start):int(cell.y_stop),
int(cell.x_start):int(cell.x_stop), 1] = 0
        parentlist.append(parent)
        callback(parent, start_squar, parentlist, im2)


    closed_list.reverse()
    parentliste = []
    ans = callback(goal_square, start_square, parentliste, im2)
    plt.imshow(im2)
    plt.show()
```