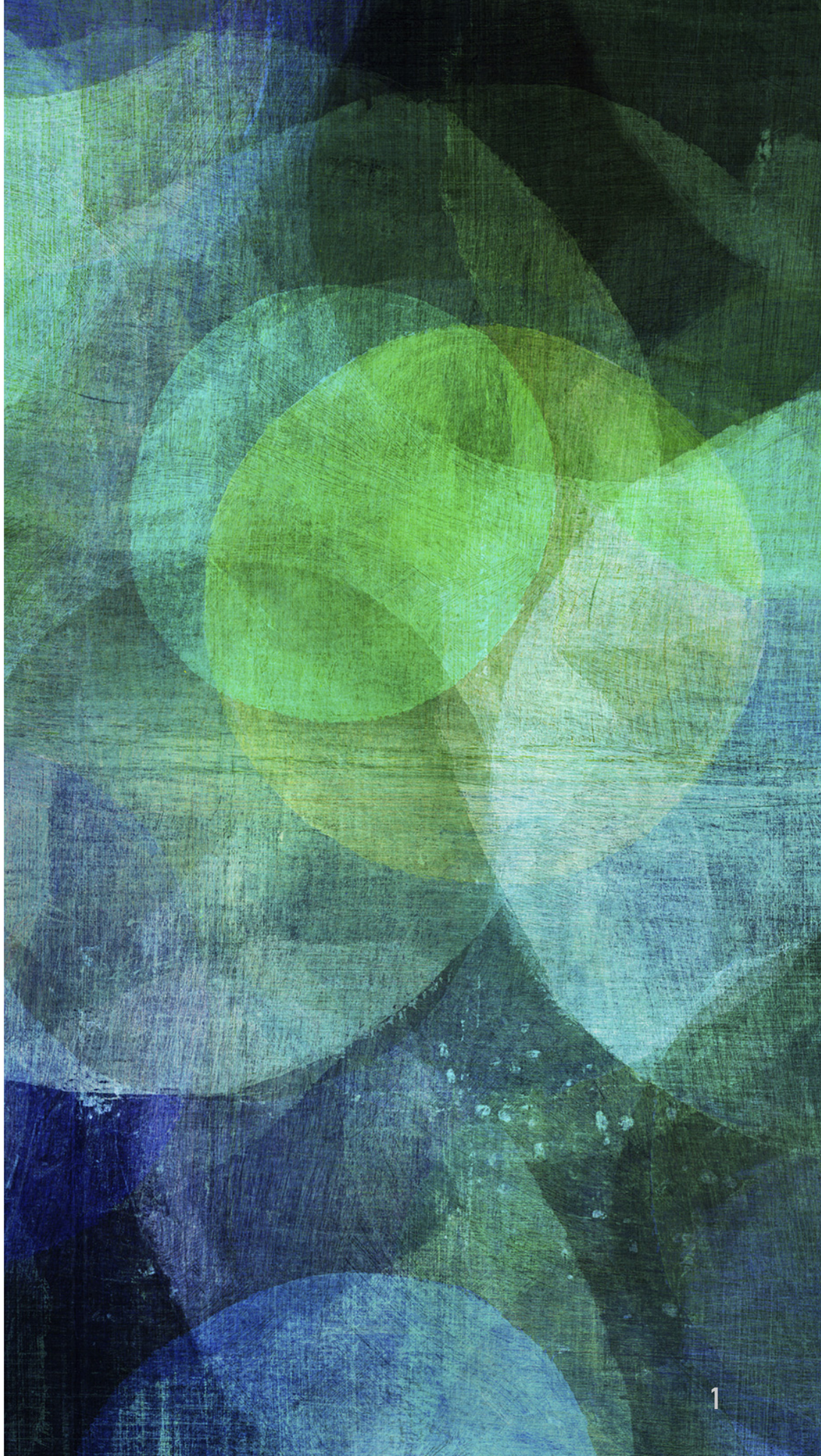
COMPARISON BETWEEN A STAR AND DIJKSTRA ALGORITHM WITH DIFFERENT KERNEL AND OBSTACLE SIZES AND HEURISTICS USING OCCUPANCY GRID IN AN AUTONOMOUS VEHICLE

Supervisor: Dr. Francesco Maurelli & Dr. Szymon Krupiński
& Arturo Gomez Chavez

By: Abrish Sabri



Project Goal

- The aim of this research is to find the shortest, smartest path between a source and a destination
  - by comparing the two algorithms
    - A star and Dijkstra
      - heuristics
- within different kernel and obstacle sizes
- Overall processing time taken
- Space and Time Complexity

Motivation

- Finds the shortest path between a source and the destination through obstacles
- How does A star and Dijkstra output differ?
- Does different kernel sizes affect the path?
- Is there a significant difference when different heuristics are used?
- Is there a significant difference in path when obstacles sizes are increased?

Importance

- Algorithms to find a shortest path are important not only in robotics, but also in video games, network routing and gene sequencing [3]

Related Work on this Task

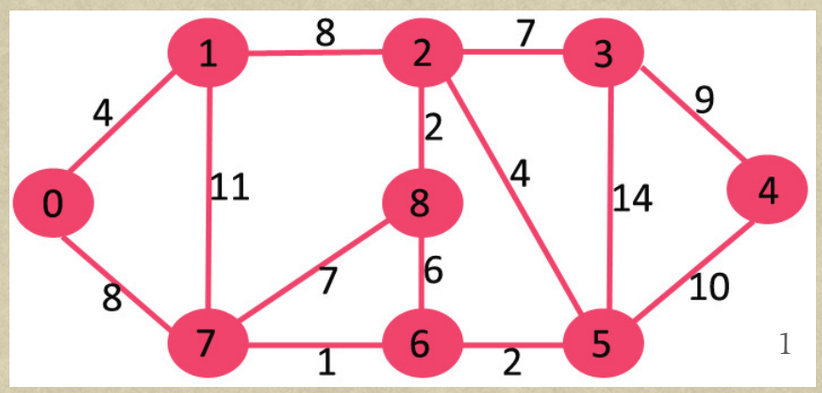
1. Introduction to A* [4]
2. On the heuristics of A* in ITS and robot path planning [6]
3. Syed Abdullah Fadzli, Sani Iyal Abdulkadir, Mokhairi Makhtar, and Azrul Amri Jamal. Robotic indoor path planning using Dijkstra's algorithm with multi-layer dictionaries. In 2015 2nd International Conference on Information Science and Security (ICISS), pages 1-4. IEEE, 2015. [7]

OCCUPANCY GRID

- An array with occupancy variables
- This method divides area into cells and categorizes as occupied, free or unknown.
- Occupancy probability
  - if the grid cell is occupied: p(mi) = 1
  - if the grid cell is not occupied: p(mi) = 0
  - no knowledge of the grid cell: p(mi) = 0.5
- Trajectory is found by computing the shortest path that do not cross any of the occupied cells

Dijkstra Algorithm

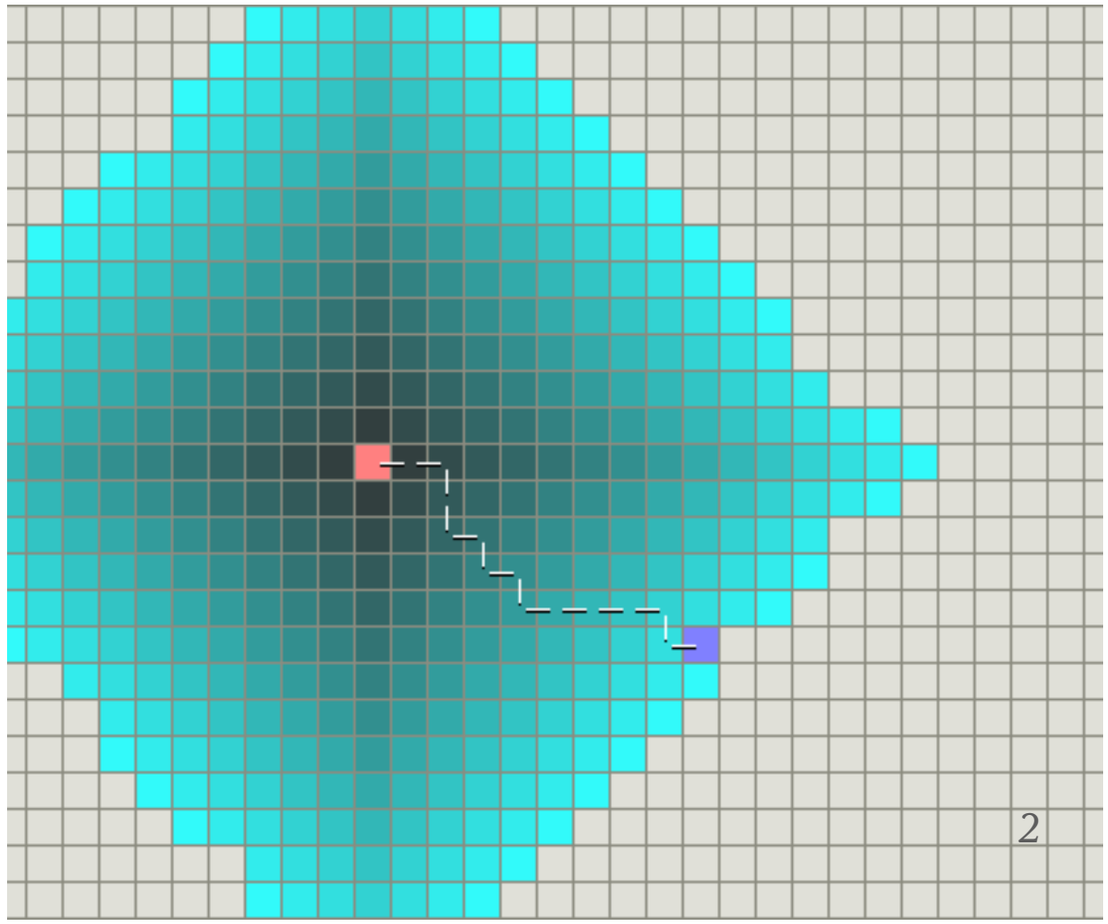
- Algorithm to calculate the shortest path from the source to all vertices in the given graph (grid)
- Invented by Edsger W. Dijkstra in 1956
- Greedy Algorithm
- Weighted Graph G = (E,V)
- Graphs must be connected
- Directed and undirected graph
- Edges must have non-negative weights
- Time Complexity; worst-case: O(V)^2
- Space Complexity; worst-case: O(V)
- Applications:
  - Geographical Maps
  - IP routing
  - Telephone network



A star Algorithm

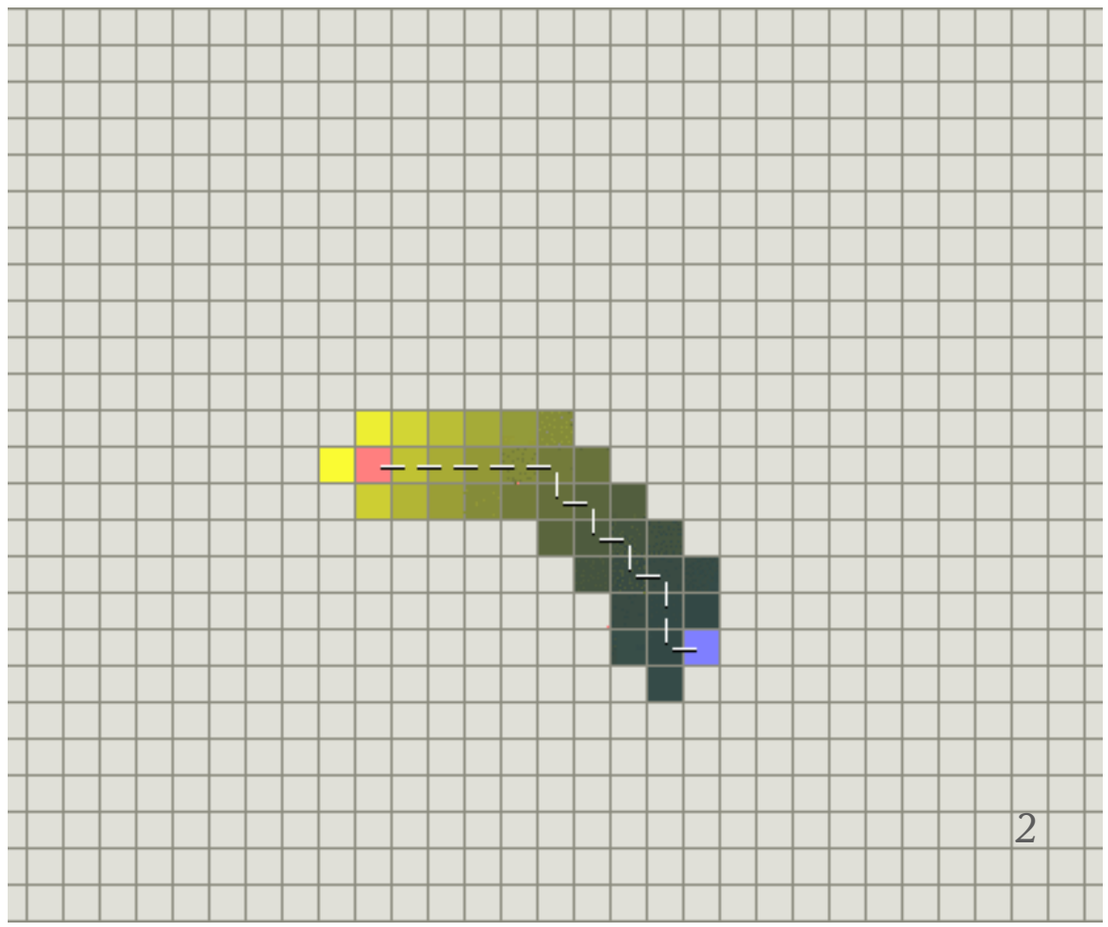
- Algorithm to calculate the shortest path from the source to a given destination.
- Extension of Dijkstra by using heuristics
- Published in 1968
- Time Complexity; worst-case: O(|E|) = O(b^d)
- Space Complexity; worst-case: O(|V|) = O(b^d)
- A star uses a function that finds an estimate of the total cost of a path for each node.
  - f(n) = g(n)+h(n)
    - f(n); total cost of path through node
    - g(n); current cost to reach node
    - h(n); estimated cost from current node to destination node
- Three types of heuristics: Manhattan, Diagonal & Euclidean

Dijkstra



Blue color shows the number of nodes visited by Dijkstra's algorithm

A Star



Yellow color means nodes closer to source node and green means nodes closer to the destination

Differences in number of vertices explore to compute the path

Dijkstra

0->122->244->366->488->610->732->854->976->1098->1220->1342->1464->1586->1708->1830->1952->2074->2196->2318->2440->2562->2684->2806->2928->3050->3172->3294->3416->3538->3660->3782->3904->4026->4148->4149->4150->4272->4273->4274->4275->4276->4398->4518->4640->4762->4884->5006->5128->5248->5368->5488->5608->5728->5848->5968->5967->5966->5844->5843->5842->5841->5840->5839->5838->5958->6080->6202->6324->6446->6568->6690->6812->6934->7056->7178->7300->7422->7544->7666->7788->7910->8032->8154->8276->8398->8399->8400->8522->8523->8524->8646->8647->8648->8649->8650->8772->8773->8774->8775->8897->8898->8899->8900->8901->9023->9024->9025->9026->9148->9149->9029->9030->9031->9153->9154->9155->9156->9278->9279->9280->9281->9403->9404->9405->9406->9407->9529->9649->9769->9889->10009->10129->10249->10369->10489->10609->10729->10849->10969->11089->11209->11329->11449->11569->11689->11688->11687->11686->11685->11684->11562->11561->11560->11559->11558->11557->11556->11434->11433->11432->11431->11430->11429->11549->11548->11668->11790->11912->12034->12156->12278->12400->12522->12644->12766->12888->13010->13132->13254->13376->13498->13620->13742->13864->13986->14108->14230->14352->14474->14596->14718->14719->14720->14721->14722->14723->14724->14725->14726->14727->14728->14729->14730->14731->14732->14733->14734->14735->14736->14737->14738->14739->14740->14741->14742->14743->14744->14745->14746->14747->14748->14749->14750->14751->14752->14753->14754->14755->14756->14757->14758->14759->14760->14761

231

0->74->148->222->296->370->444->518->592->666->740->814->888->962->890->818->892->893->894->895->896->897->971->1045->1046->1120->1121->1122->1196->1197->1271->1345->1419->1493->1567->1640->1713->1786->1860->1933->2007->2081->2155->2229->2303->2304->2378->2379->2307->2308->2309->2310->2238->2166->2094->2095->2023->2024->1952->1880->1807->1733->1732->1731->1803

65

0->53->106->159->212->265->318->371->424->477->426->479->428->429->430->431->484->537->590->591->592->593->646->699->751->804->856->908->960->1013->1066->1119->1172->1173->1174->1175->1176->1177->1126->1075->1024->972->920->869->817->766->714->663->610->557->506->454->402

53

A Star

(10,10) -> (11,10) -> (12,10) -> (13,10) -> (14,10) -> (15,10) -> (16,10) -> (17,10) -> (18,10) -> (19,10) -> (20,10) -> (21,10) -> (22,11) -> (23,12) -> (24,13) -> (25,13) -> (26,13) -> (27,13) -> (28,13) -> (29,13) -> (30,13) -> (31,13) -> (32,13) -> (33,14) -> (34,15) -> (35,16) -> (36,17) -> (37,18) -> (38,19) -> (39,20) -> (40,21) -> (41,22) -> (42,23) -> (43,24) -> (44,25) -> (45,26) -> (46,27) -> (47,28) -> (48,29) -> (49,30) -> (50,31) -> (51,32) -> (52,33) -> (53,34) -> (54,35) -> (55,36) -> (56,37) -> (57,38) -> (58,39) -> (59,40) -> (60,41) -> (61,42) -> (62,43) -> (63,44) -> (64,45) -> (65,46) -> (66,47) -> (67,48) -> (68,49) -> (69,50) -> (70,51) -> (71,52) -> (72,53) -> (73,54) -> (74,55) -> (75,56) -> (76,57) -> (77,58) -> (78,59) -> (79,60) -> (80,61) -> (80,62) -> (80,63) -> (80,64) -> (80,65) -> (80,66) -> (80,67) -> (80,68) -> (80,69) -> (80,70) -> (80,71) -> (80,72) -> (80,73) -> (80,74) -> (80,75) -> (80,76) -> (80,77) -> (80,78) -> (80,79) -> (80,80) -> (81,81) -> (82,82) -> (83,83) -> (84,84) -> (85,85) -> (86,86) -> (87,87) -> (88,88) -> (89,89) -> (90,90) -> (91,91) -> (92,92) -> (93,93) -> (94,94) -> (95,95) -> (96,96) -> (97,97) -> (98,98) -> (99,99) -> (100,100) -> (101,101) -> (102,102) -> (103,103) -> (104,104) -> (105,105) -> (106,106) -> (107,107) -> (108,108) -> (109,109) -> (110,110) -> (111,111) -> (112,112) -> (113,113) -> (114,114) -> (115,115) -> (116,116) -> (117,117) -> (118,118) -> (119,119) -> (120,120)

108

(2,2) -> (3,2) -> (4,2) -> (5,2) -> (6,2) -> (7,2) -> (8,2) -> (9,2) -> (10,2) -> (11,2) -> (12,2) -> (13,2) -> (14,3) -> (15,4) -> (16,5) -> (17,5) -> (18,6) -> (19,7) -> (20,8) -> (21,9) -> (22,10) -> (23,11) -> (24,12) -> (24,13) -> (24,14) -> (24,15) -> (24,16) -> (24,17) -> (24,18) -> (24,19) -> (24,20) -> (24,21) -> (24,22) -> (24,23) -> (24,24)

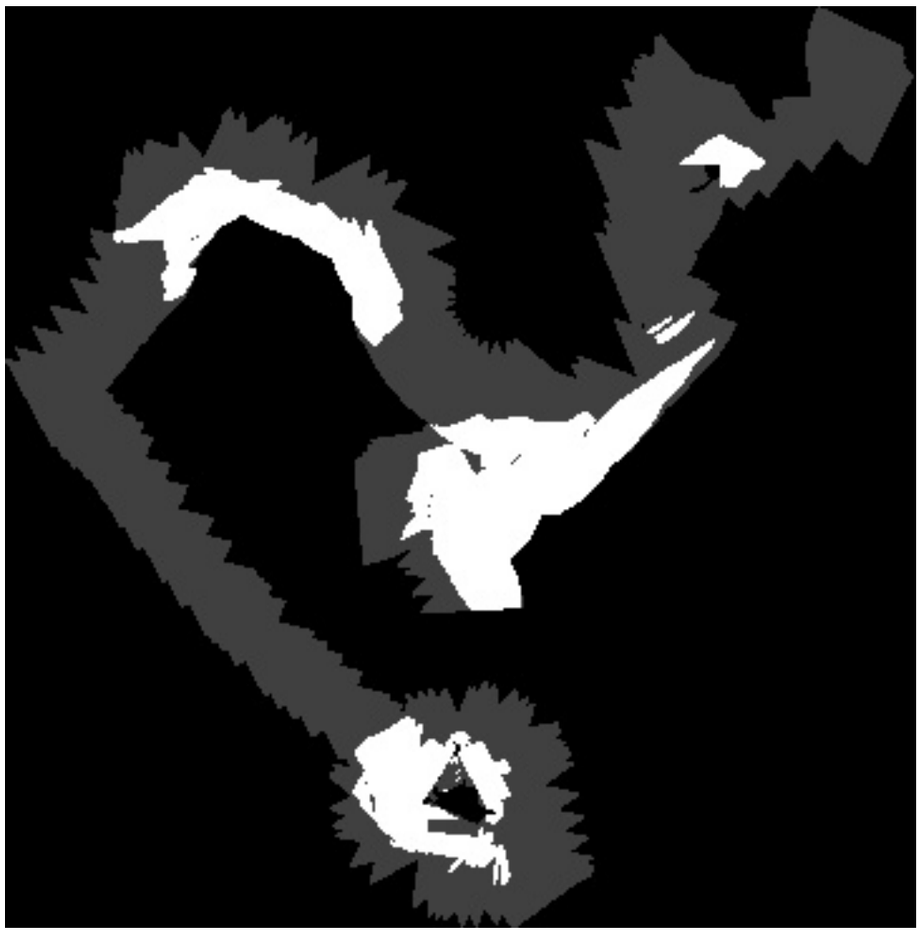
36

(1,1) -> (2,1) -> (3,1) -> (4,1) -> (5,1) -> (6,1) -> (7,1) -> (8,1) -> (9,1) -> (10,2) -> (11,3) -> (12,4) -> (13,5) -> (14,6) -> (15,7) -> (16,8) -> (17,9) -> (17,10) -> (17,11) -> (17,12) -> (17,13) -> (17,14) -> (17,15) -> (17,16) -> (17,17)

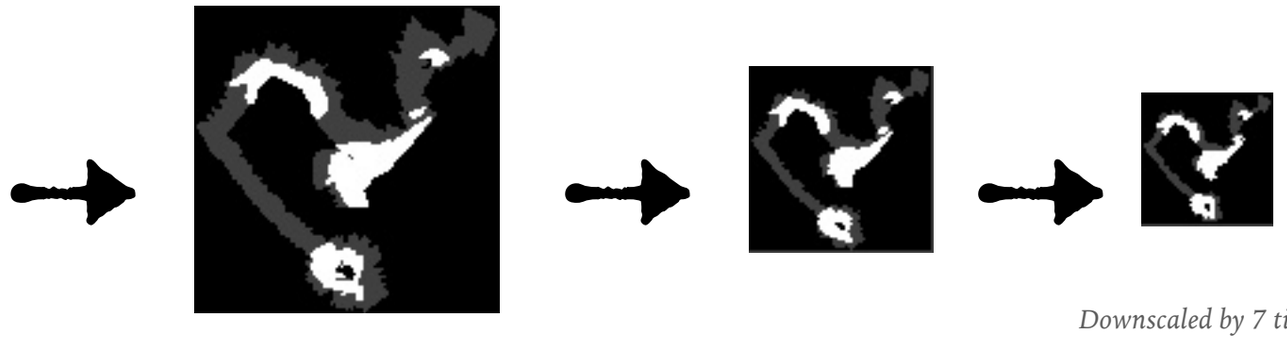
25

data recorded of downsample3.txt, downsample5.txt & downsample7.txt

Downsampled Images



Original Map

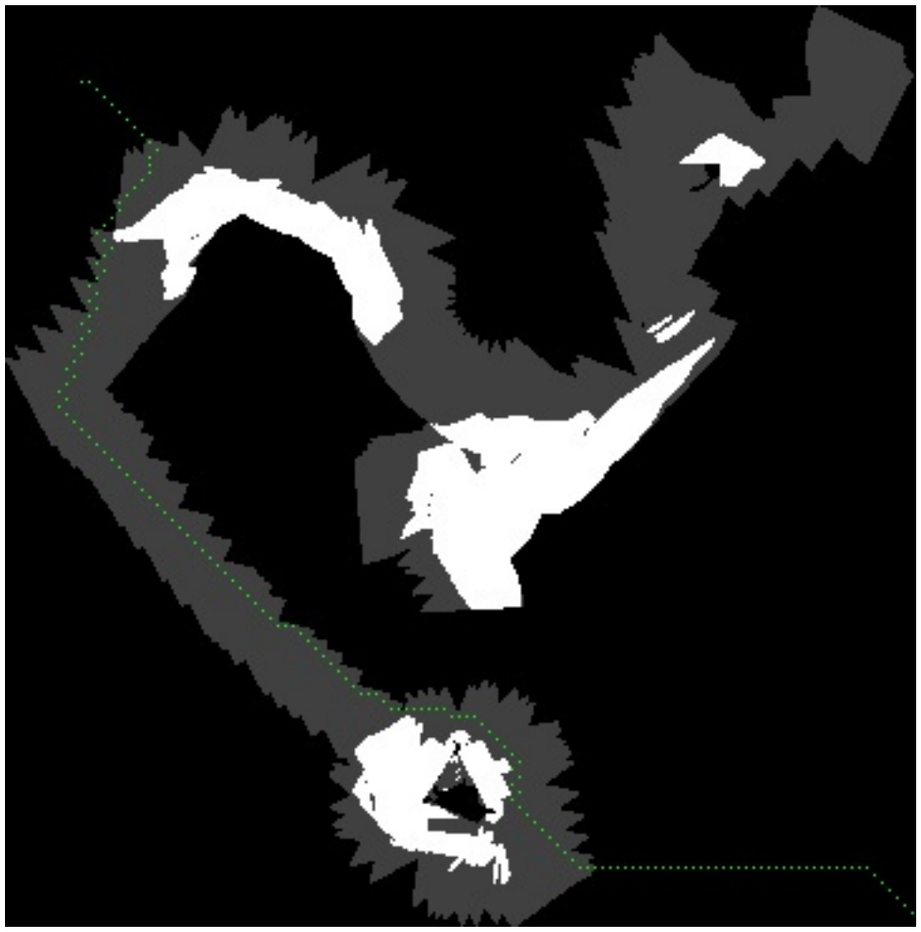


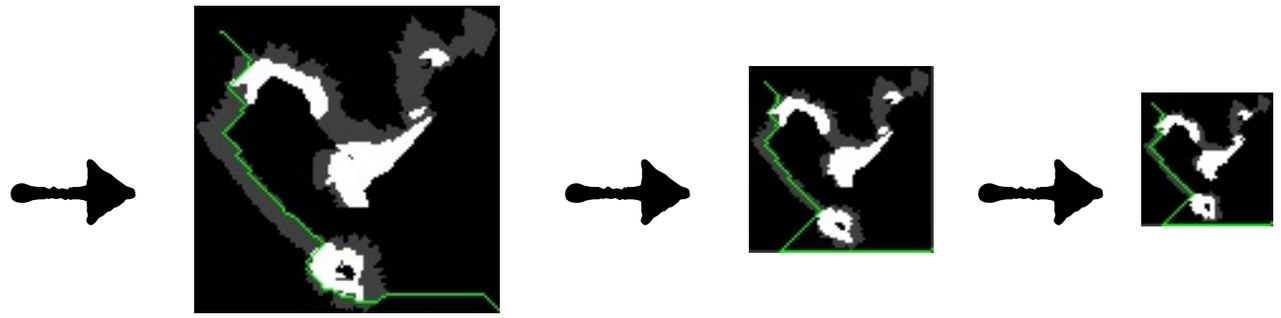
Downscaled by 3 times

Downscaled by 5 times

Downscaled by 7 times

All Images are downsampled by using the convolution method with filter2D





Same path downscaled by 7 times

Same path downscaled by 5 times

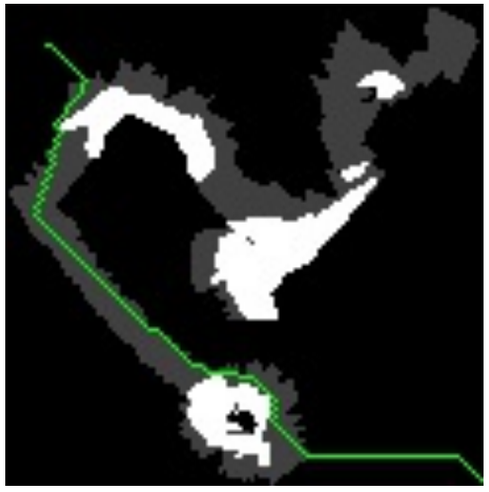
Same path downscaled by 3 times

A star computed on original map

DIFFERENCES IN PATH WITH DIFFERENT KERNEL SIZES IN A STAR

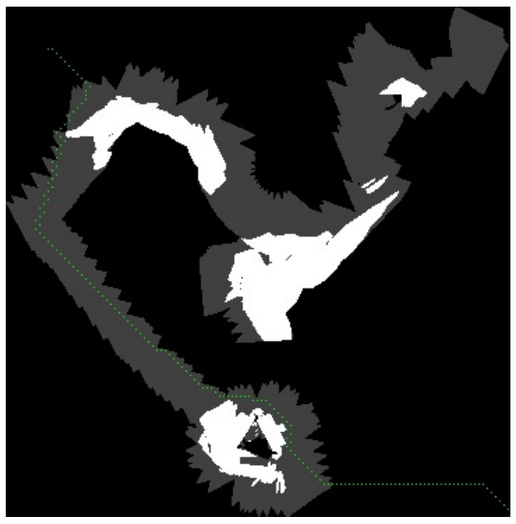
Downscaled image

Kernel size : 3



Path Computed by A star

Source: (30,30) Destination: (360,360)



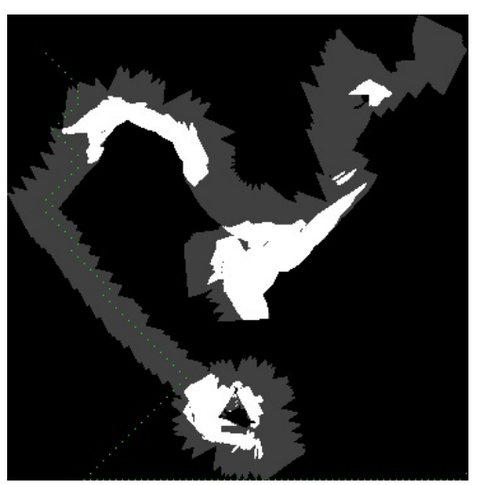
Correct Path after computing the correct source and destination points within the right resolution : Source: (10,10) Destination: (120,120)

Downscaled image

Kernel size : 5

Source: (6,6) Destination: (72,72)

Path differs due to grid size differences



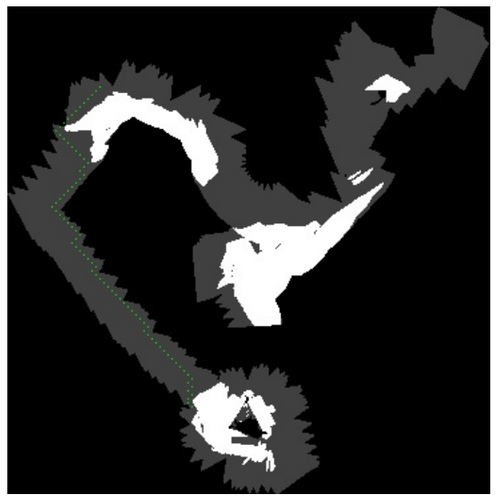
Correct Path:

Source (0,0) -> (6,6)

Destination: (120,120) -> (72,72)

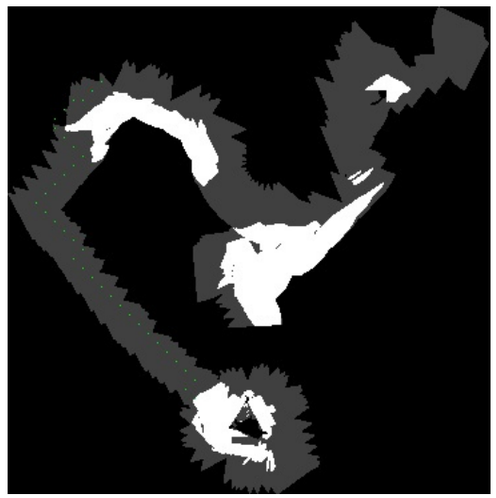
Differences in Path with different heuristics in A star

Manhattan:



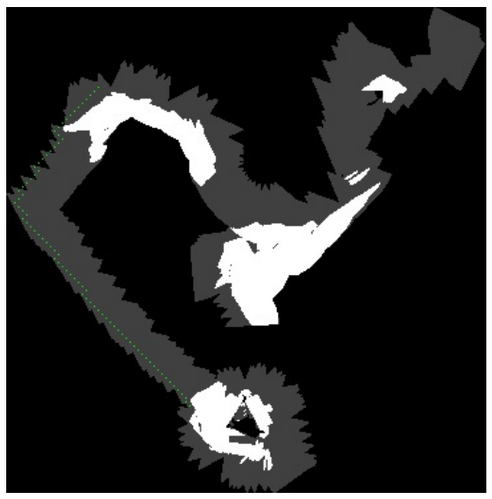
Kernel Size 3

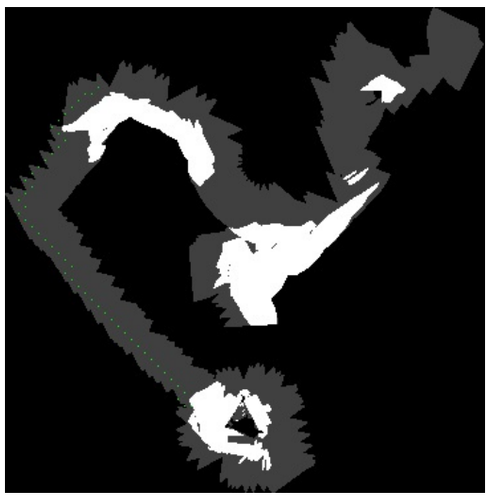
Kernel Size 5

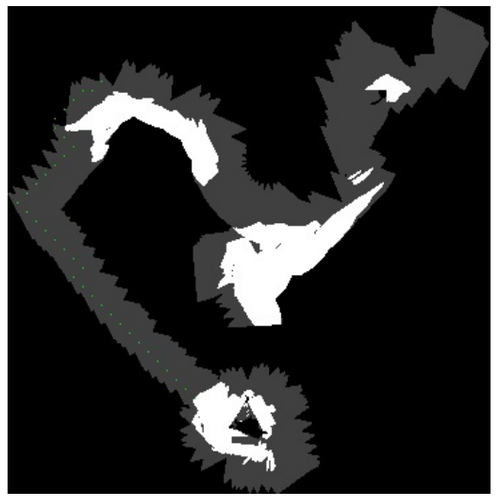


Kernel Size 7

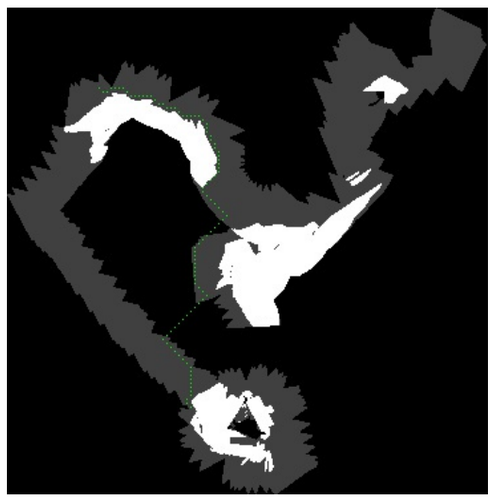
Euclidean:

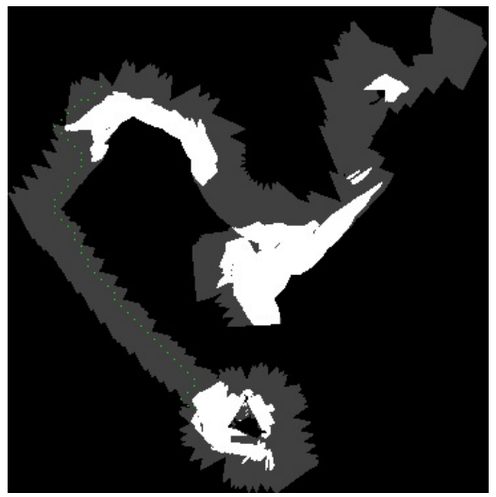






Diagonal:

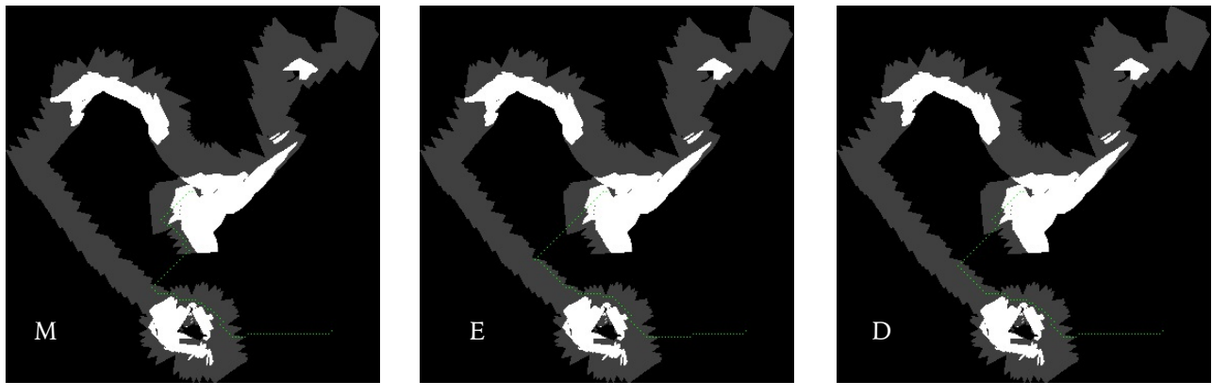




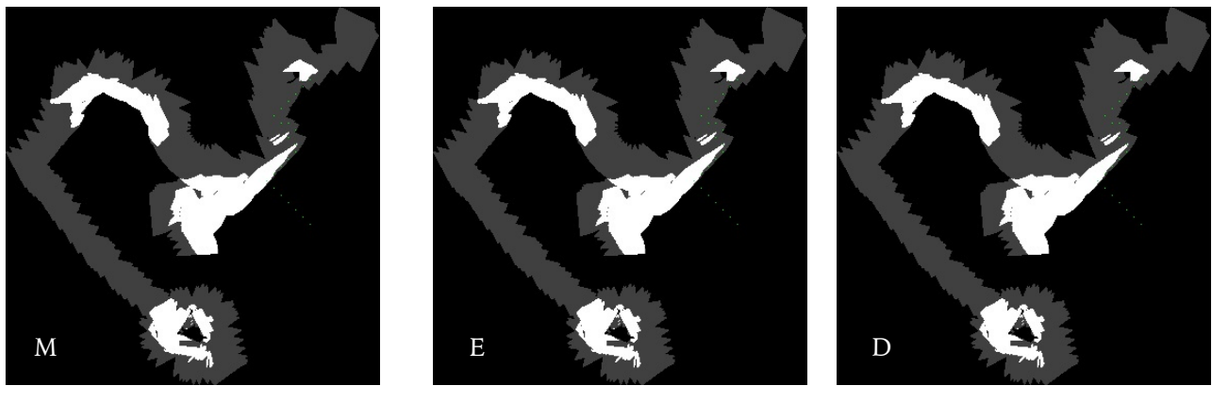
Comparison with all three Heuristics

Compared these heuristics with 10 sample coordinates in different kernel sizes

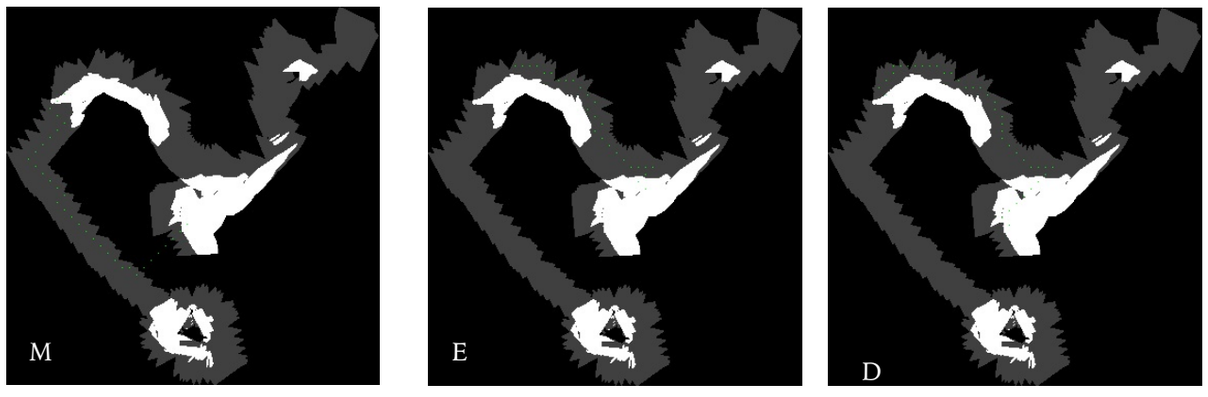
1. Source: 30,30 Destination: 360,360
2. Source: 60,70 Destination: 60,340
3. Source: 60,70 Destination: 140,300
4. Source: 60,90 Destination: 180,210
5. Source: 60,60 Destination: 180,315
6. Source: 300,70 Destination: 300,210
7. Source: 300,70 Destination: 180,315
8. Source: 180,180 Destination: 315,315
9. Source: 40,320 Destination: 340,60
10. Source: 40,320 Destination: 300,320



Path computed with coordinates no 8 in kernel size 3



Path computed with coordinates no 6 in kernel size 7



Path computed with coordinates no 4 in kernel size 5

Comparison with different Obstacle Sizes I

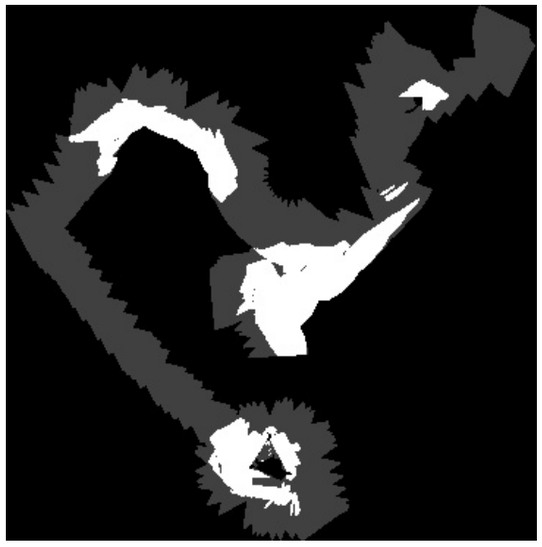


Fig:1 100,50,1

Fig:2 100,2,1

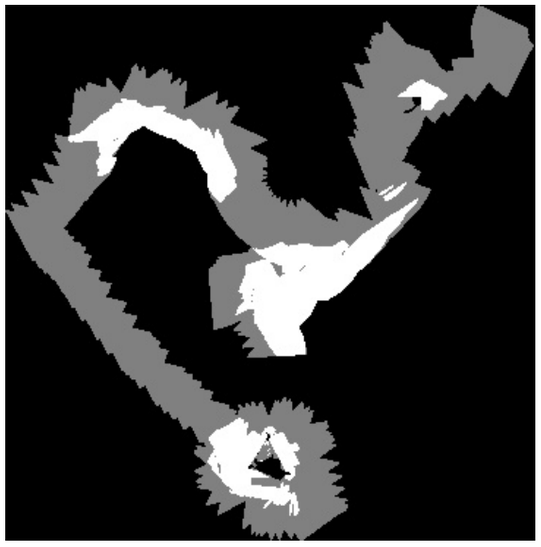


Fig:3 60,50,40

The free cells in fig 2 is barely visible

In fig 3, the free cells have a lighter grey color closer to white.

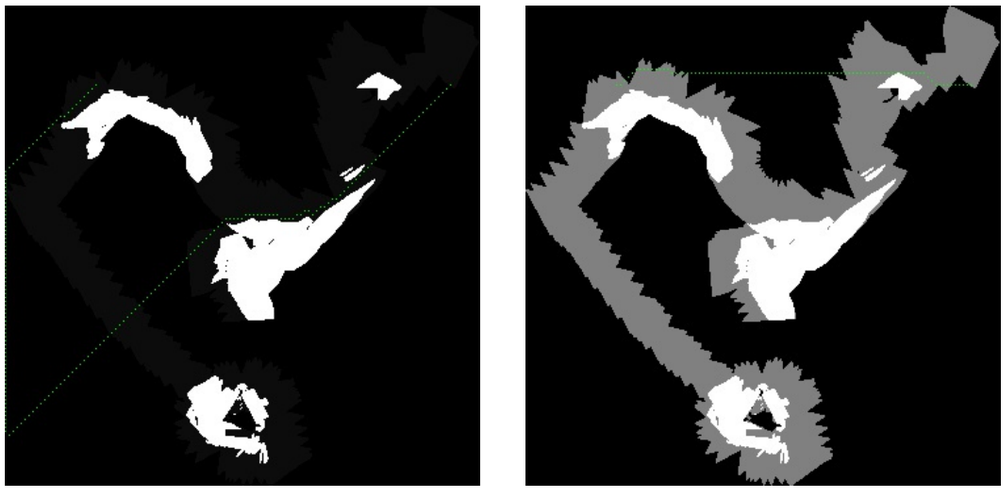
Comparison with different Obstacle Sizes II

Source : 30,30

Destination: 360,360

Source: 60,70

Destination: 60,340



- By changing the values, the path changes completely
- There is almost no recognization of free cells in first set
- In second set, the unknown cells are mistaken with free cells and computes the shortest path by making a straight line and passes through the obstacles respectively.

Path Comparison

Kernel Size 3

Euclidean
(30,20) -> (31,19) -> (32,18) -> (33,17) -> (34,16) -> (33,15) -> (32,14) -> (31,13) -> (30,13) -> (29,14) -> (28,15) -> (28,16) -> (27,17) -> (26,18) -> (26,19) -> (25,20) -> (24,21) -> (23,22) -> (22,23) -> (21,24) -> (21,25) -> (20,26) -> (20,27) -> (20,28) -> (20,29) -> (21,30) -> (22,31) -> (22,32) -> (22,33) -> (22,34) -> (22,35) -> (22,36) -> (23,37) -> (24,38) -> (24,39) -> (25,40) -> (25,41) -> (25,42) -> (26,43) -> (27,44) -> (27,45) -> (27,46) -> (27,47) -> (27,48) -> (28,49) -> (29,50) -> (30,50) -> (31,50) -> (32,51) -> (33,51) -> (34,52) -> (35,52) -> (36,53) -> (37,53) -> (38,54) -> (39,55) -> (40,56) -> (41,57) -> (42,58) -> (43,59) -> (44,60) -> (45,61) -> (46,62) -> (47,63) -> (48,64) -> (49,65) -> (50,66) -> (51,67) -> (52,68) -> (52,69) -> (52,70) -> (52,71) -> (52,72) -> (52,73) -> (52,74) -> (52,75) -> (52,76) -> (52,77) -> (53,76) -> (54,75) -> (55,74) -> (56,73) -> (57,72) -> (58,71) -> (59,70) -> (60,69) -> (61,68) -> (62,67) -> (63,66) -> (64,65) -> (65,64) -> (66,63) -> (67,62) -> (68,61) -> (69,60) -> (70,60) = 96

Manhattan
(30,20) -> (31,19) -> (32,18) -> (33,17) -> (34,16) -> (33,15) -> (32,14) -> (31,13) -> (30,13) -> (29,14) -> (28,15) -> (28,16) -> (27,17) -> (26,18) -> (26,19) -> (25,20) -> (24,21) -> (23,22) -> (22,23) -> (21,24) -> (21,25) -> (20,26) -> (20,27) -> (20,28) -> (20,29) -> (21,30) -> (22,31) -> (22,32) -> (22,33) -> (22,34) -> (22,35) -> (22,36) -> (23,37) -> (24,38) -> (24,39) -> (25,40) -> (25,41) -> (25,42) -> (26,43) -> (27,44) -> (27,45) -> (27,46) -> (27,47) -> (27,48) -> (28,49) -> (29,50) -> (30,51) -> (31,52) -> (32,53) -> (33,54) -> (34,55) -> (35,56) -> (36,57) -> (37,56) -> (38,57) -> (39,56) -> (40,57) -> (41,58) -> (42,59) -> (43,58) -> (44,59) -> (45,60) -> (46,61) -> (47,62) -> (48,63) -> (49,64) -> (50,65) -> (51,66) -> (52,67) -> (52,68) -> (52,69) -> (52,70) -> (52,71) -> (52,72) -> (52,73) -> (52,74) -> (52,75) -> (52,76) -> (52,77) -> (53,76) -> (54,75) -> (55,74) -> (56,73) -> (57,72) -> (58,71) -> (59,70) -> (60,69) -> (61,68) -> (62,67) -> (63,66) -> (64,65) -> (65,64) -> (66,63) -> (67,62) -> (68,61) -> (69,60) -> (70,60) = 97

Diagonal
(30,20) -> (31,19) -> (32,18) -> (33,17) -> (34,16) -> (33,15) -> (32,14) -> (31,13) -> (30,13) -> (29,14) -> (28,15) -> (28,16) -> (27,17) -> (26,18) -> (26,19) -> (25,20) -> (24,21) -> (23,22) -> (22,23) -> (21,24) -> (21,25) -> (20,26) -> (20,27) -> (20,28) -> (20,29) -> (21,30) -> (22,31) -> (22,32) -> (22,33) -> (22,34) -> (22,35) -> (22,36) -> (23,37) -> (24,38) -> (24,39) -> (25,40) -> (25,41) -> (25,42) -> (26,43) -> (27,44) -> (27,45) -> (27,46) -> (27,47) -> (27,48) -> (28,49) -> (29,50) -> (30,51) -> (31,52) -> (32,53) -> (33,54) -> (34,55) -> (35,56) -> (36,57) -> (37,56) -> (38,57) -> (39,56) -> (40,57) -> (41,58) -> (42,59) -> (43,58) -> (44,59) -> (45,60) -> (46,61) -> (47,62) -> (48,63) -> (49,64) -> (50,65) -> (51,66) -> (52,67) -> (52,68) -> (52,69) -> (52,70) -> (52,71) -> (52,72) -> (52,73) -> (52,74) -> (52,75) -> (52,76) -> (52,77) -> (53,76) -> (54,75) -> (55,74) -> (56,73) -> (57,72) -> (58,71) -> (59,70) -> (60,69) -> (61,68) -> (62,67) -> (63,66) -> (64,65) -> (65,64) -> (66,63) -> (67,62) -> (68,61) -> (69,60) -> (70,60) = 97

Kernel Size 7

Euclidean
(42,10) -> (43,9) -> (44,8) -> (45,7) -> (46,6) -> (47,5) -> (48,4) -> (49,3) -> (50,2) -> (51,1) -> (52,0) -> (52,1) -> (52,2) -> (52,3) -> (52,4) -> (52,5) -> (52,6) -> (52,7) -> (52,8) -> (52,9) -> (52,10) -> (52,11) -> (52,12) -> (52,13) -> (52,14) -> (52,15) -> (52,16) -> (52,17) -> (52,18) -> (52,19) -> (52,20) -> (52,21) -> (52,22) -> (52,23) -> (52,24) -> (52,25) -> (52,26) -> (52,27) -> (52,28) -> (51,28) -> (50,28) -> (49,29) -> (48,29) -> (47,29) -> (46,29) -> (45,29) -> (44,29) -> (43,29) -> (42,29) -> (41,28) -> (40,27) -> (40,26) -> (40,25) -> (40,24) -> (39,24) -> (38,24) -> (37,24) -> (36,24) -> (35,24) -> (34,24) -> (33,24) -> (32,24) -> (31,23) -> (30,23) -> (31,22) -> (30,21) -> (29,21) -> (28,21) -> (27,21) -> (26,22) -> (25,22) -> (24,22) -> (23,23) -> (22,24) -> (22,25) -> (22,26) -> (22,27) -> (22,28) -> (22,29) -> (22,30) -> (22,31) -> (22,32) -> (21,33) -> (21,34) -> (20,35) -> (20,36) -> (19,36) -> (18,35) -> (17,36) -> (16,37) -> (15,37) -> (14,37) -> (13,37) -> (12,37) -> (11,37) -> (10,37) -> (9,37) -> (8,37) -> (7,38) -> (6,39) -> (6,40) -> (6,41) -> (7,42) -> (7,43) -> (7,44) -> (7,45) -> (6,46) -> (6,47) -> (7,48) -> (8,48) = 110

Manhattan
(42,10) -> (43,9) -> (44,8) -> (45,7) -> (46,6) -> (47,5) -> (48,4) -> (49,3) -> (50,2) -> (51,1) -> (52,0) -> (52,1) -> (52,2) -> (52,3) -> (52,4) -> (52,5) -> (52,6) -> (52,7) -> (52,8) -> (52,9) -> (52,10) -> (52,11) -> (52,12) -> (52,13) -> (52,14) -> (52,15) -> (52,16) -> (52,17) -> (52,18) -> (52,19) -> (52,20) -> (52,21) -> (52,22) -> (52,23) -> (52,24) -> (52,25) -> (52,26) -> (52,27) -> (52,28) -> (51,28) -> (50,28) -> (49,29) -> (48,29) -> (47,29) -> (46,29) -> (45,29) -> (44,29) -> (43,29) -> (42,29) -> (41,29) -> (40,28) -> (39,27) -> (40,26) -> (40,25) -> (40,24) -> (39,24) -> (38,24) -> (37,24) -> (36,24) -> (35,24) -> (34,24) -> (33,24) -> (32,24) -> (31,23) -> (30,23) -> (31,22) -> (30,21) -> (29,21) -> (28,21) -> (27,21) -> (26,22) -> (25,22) -> (24,22) -> (23,23) -> (22,24) -> (22,25) -> (22,26) -> (22,27) -> (22,28) -> (22,29) -> (22,30) -> (22,31) -> (22,32) -> (21,33) -> (21,34) -> (20,35) -> (20,36) -> (19,36) -> (18,35) -> (17,36) -> (16,37) -> (15,37) -> (14,37) -> (13,37) -> (12,37) -> (11,37) -> (10,37) -> (9,37) -> (8,37) -> (7,38) -> (6,39) ->(6,40) -> (6,41) -> (7,42) -> (7,43) -> (7,44) -> (7,45) -> (6,46) -> (6,47) -> (7,48) -> (8,48) = 111

Diagonal:
(42,10) -> (43,9) -> (44,8) -> (45,7) -> (46,6) -> (47,5) -> (48,4) -> (49,3) -> (50,2) -> (51,1) -> (52,0) -> (52,1) -> (52,2) -> (52,3) -> (52,4) -> (52,5) -> (52,6) -> (52,7) -> (52,8) -> (52,9) -> (52,10) -> (52,11) -> (52,12) -> (52,13) -> (52,14) -> (52,15) -> (52,16) -> (52,17) -> (52,18) -> (52,19) -> (52,20) -> (52,21) -> (52,22) -> (52,23) -> (52,24) -> (52,25) -> (52,26) -> (52,27) -> (52,28) -> (52,29) -> (51,29) -> (50,29) -> (49,29) -> (48,29) -> (47,29) -> (46,29) -> (45,29) -> (44,29) -> (43,29) -> (42,29) -> (41,29) -> (40,28) -> (39,27) -> (40,26) -> (40,25) -> (40,24) -> (39,24) -> (38,24) -> (37,24) -> (36,24) -> (35,24) -> (34,24) -> (33,24) -> (32,24) -> (31,23) -> (30,23) -> (31,22) -> (30,21) -> (29,21) -> (28,21) -> (27,21) -> (26,22) -> (25,22) -> (24,22) -> (23,23) -> (22,24) -> (22,25) -> (22,26) -> (22,27) -> (22,28) -> (22,29) -> (22,30) -> (22,31) -> (22,32) -> (21,33) -> (21,34) -> (20,35) -> (20,36) -> (19,36) -> (18,35) -> (17,36) -> (16,37) -> (15,37) -> (14,38) -> (13,38) -> (12,39) -> (11,38) -> (10,37) -> (9,38) -> (8,37) -> (7,38) -> (6,39) -> (6,40) -> (6,41) -> (7,42) -> (7,43) -> (7,44) -> (7,45) -> (6,46) -> (6,47) -> (7,48) -> (8,48) = 112

Kernel Size 7

Euclidean
(12,8) -> (13,7) -> (14,6) -> (15,7) -> (16,8) -> (17,7) -> (18,6) -> (19,5) -> (20,4) -> (21,3) -> (22,4) -> (23,5) -> (24,6) -> (25,7) -> (26,8) -> (27,7) ->(28,8) -> (29,9) -> (30,10) -> (31,11) -> (32,12) -> (33,13) -> (34,14) -> (35,15) ->(36,16) -> (36,17) ->(37,18) ->(36,19)->(35,20) -> (34,21) ->(33,22)-> (32,23) -> (31,24) -> (30,25) = 34

Manhattan
(12,8) -> (11,7) -> (10,8) -> (9,9) -> (8,9) -> (8,10) -> (8,11) -> (8,12) ->(8,13) -> (8,14) -> (8,15) -> (9,16) -> (9,17) -> (10,18) -> (11,19) -> (11,20) ->(11,21) -> (12,22) -> (13,23) -> (14,22) -> (15,23) -> (16,24) -> (17,23) -> (18,24) ->(19,25) -> (20,26) ->(21,27) -> (22,28) -> (22,29) -> 22,30) ->(22,31) -> (23,30) ->(24,29) ->(25,28)-> (26,27) -> (27,26) ->(28,25)-> (29,24) -> (30,25) = 39

Diagonal
(12,8) -> (11,7) -> (10,8) -> (9,9) -> (8,9) ->(8,10) ->(8,11) ->(8,12) ->(8,13) -> (8,14) -> (8,15) -> (9,16) -> (9,17) -> (10,18) -> (11,19) -> (11,20) ->(11,21) -> (12,22) -> (13,23) -> (14,23) ->(15,24)-> (16,24) -> (17,24) ->(18,24) ->(19,25) -> (20,26)->(21,27) -> (22,28) ->(22,29) ->(22,30) -> (22,31) -> (23,30)->(24,29) -> (25,28) -> 26,27) -> (27,26) -> (28,25) -> (29,24) -> (30,25) = 39

- Different heuristics compute different paths.
- In some cases, the path is different as the kernel size increases
- In other cases, two heuristics give the shortest path such as in coordinates 3,5,6,7,8 and 9.
- Euclidean computes the shortest path 50% of the time.
- Manhattan 33%
- Diagonal 17%

Time & Space Complexity

Time Complexity

Dijkstra function : O(V^2)

A star function: O(E)

Dijkstra Program file: O(H*WxH*W)^2

A star Program file: O(H*WxH*W)^2

Space Complexity

Dijkstra: O(H*W)

A star: O(H*W)

Algorithm	Kernel Size	Heuristic	Time
Dijkstra	3	-	5.8 s
Dijkstra	5	-	4.2 s
Dijkstra	7	-	2.9 s
A star	3	euclidean	1 s
A star	3	manhattan	93ms
A star	3	diagonal	83ms
A star	5	euclidean	30 ms
A star	5	manhattan	26 ms
A star	5	diagonal	26 ms
A star	7	euclidean	26 ms
A star	7	manhattan	20 ms
A star	7	diagonal	19 ms

Table 1: Total time taken to compute paths with different heuristics and kernel sizes

Conclusion

- Dijkstra has better space complexity than A star but it may not find the optimal solution
- Although the time complexity is the same for both algorithms,
- A star calculates the shortest path much faster than Dijkstra.
- A star computes the optimal path with kernel size 3 with the euclidean heuristic function.
- The values assigned to various objects should be optimum otherwise a confused path will be evaluated.
- Kernel sizes are crucial, less resolution and data on the image and the graph returns altered path as less resolution disturbs the significance of mixed objects defined.

Libraries & outside code used 1

```
#include <cfloat>
#include <cmath>
#include <fstream>
#include <iostream>
#include <set>
#include <sstream>
#include <stack>
#include <string>
#include <utility>
#include <vector>
#include <stdlib.h>
#include <stdio.h>
#include <iomanip>
#include <limits.h>
#include "yaml-cpp/yaml.h"
#include "opencv2/imgproc/imgproc.hpp"
#include “opencv2/highgui/highgui.hpp"
```

- roslaunch dexrov_meta startup_simulartion.launch ; starts up the simulation
- roslaunch dexrov_meta startup_vehicle_sim.launch ; vehicle is imported in the simulation
- roslaunch dexrov_meta startup_vessel_sim.launch ; vessel is imported in the simulation
- roslaunch dexrov_meta rviz.launch ; for launching rviz and visualization purpose to keep track of all the topics and nodes
- roslaunch dexrov_meta perception_octomap.launch ; for launching the octomap
- rosrun joy joy_node _dev:=/dev/input/js1 ; in case the joystick does not work

```
if (cellDetails[i - 1][j + 1].f == FLT_MAX || cellDetails[i - 1][j + 1].f > fNew)
{openList.insert(make_pair(fNew, make_pair(i - 1, j + 1)));
cellDetails[i - 1][j + 1].f = fNew;
cellDetails[i - 1][j + 1].g = gNew;
cellDetails[i - 1][j + 1].h = hNew;
cellDetails[i - 1][j + 1].parent_i = i;
cellDetails[i - 1][j + 1].parent_j = j; }
```

code snippet from astar.cpp for updating the values of f and g.

```
kernel = Mat::ones( kernel_size, kernel_size, CV_32FC1 )/ (float)(kernel_size*kernel_size);
cv::filter2D(M, dst, ddepth , kernel, anchor, delta, BORDER_DEFAULT );
```

code snippet from filter2D.cpp

```
g++ filter2D.cpp -o filter2D `pkg-config --cflags --libs opencv`
g++  -std=c++11 astar.cpp -o astar -lyaml-cpp `pkg-config --cflags --libs opencv`
g++  -std=c++11 dij.cpp -o dij -lyaml-cpp `pkg-config --cflags --libs opencv`
```

Boundaries of this thesis

- SIGKILL 9 ; 132492 vector
- Out of Memory
- Tie Breaking Mechanism
- Filter2D convolution

$$dst(x,y) = \sum_{i=0}^{M_i-1} \sum_{j=0}^{M_j-1} kernel(i,j) * src(x+i-anchor_i, y+j-anchor_j)$$

Possible Improvements

- Data Structure
- Heuristics
- Map Representations
- Variety of Map Images

REFERENCES

1. “Dijsktra's Algorithm” *GeeksforGeeks*, 4 July 2017, www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/.
2. “A* Search Algorithm - Mohit Joshi.” *HackerEarth*, www.hackerearth.com/practice/notes/a-search-algorithm/.
3. ‘Introduction to Robotics #4: Path-Planning.” Correll Lab, 14 Oct. 2014, correll.cs.colorado.edu/?p=965.
4. “Introduction to A*.” *Red Blob Games*, www.redblobgames.com/pathfinding/a-star/introduction.html.
5. Singh, Yogang, et al. “Towards Use of Dijkstra Algorithm for Optimal Navigation of an Unmanned Surface Vehicle in a Real-Time Marine Environment with Results from Artificial Potential Field.” *TransNav, the International Journal on Marine Navigation and Safety of Sea Transportation*, vol. 12, no. 1, 2018, doi:10.12716/1001.12.01.14.
6. T. Goto, T. Kosaka and H. Noborio, "On the heuristics of A* or A algorithm in ITS and robot path-planning," Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453), Las Vegas, NV, USA, 2003, pp. 1159-1166 vol.2. doi: 10.1109/IROS.2003.1248802 keywords: {path planning;search problems;manipulators;automobiles;graph theory;A algorithm;robot path-planning;near-optimal heuristic;A* algorithm;data sets;Dijkstra algorithms;optimal path;Path planning;Cost function;Euclidean distance;Heuristic algorithms;Intelligent transportation systems;Intelligent robots;Roads;Orbital robotics;Robotic assembly;Manipulators}, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1248802&isnumber=27959
7. Andrew B Walker et al. Hard real-time motion planning for autonomous vehicles. Walker, A. “Hard Real-Time Motion Planning for Autonomous Vehicles” PhD thesis, Swinburne University, 2011.
8. Syed Abdullah Fadzli, Sani Iyal Abdulkadir, Mokhairi Makhtar, and Azrul Amri Jamal. Robotic indoor path planning using dijkstra’s algorithm with multi-layer dictionaries. In 2015 2nd International Conference on Information Science and Security (ICISS), pages 1-4. IEEE, 2015.
9. Lisa Velden. An informed search for the shortest path.
10. Liang Yang, Juntong Qi, Dalei Song, Jizhong Xiao, Jianda Han, and Yong Xia. Survey of robot 3d path planning algorithms. Journal of Control Science and Engineering, 2016:5, 2016.
11. Dmitry S Yershov and Steven M LaValle. Simplicial dijkstra and a* algorithms for optimal feedback planning. In 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 3862-3867. IEEE, 2011.
12. Abhishek Goyal, Prateek Mogha, Rishabh Luthra, and Neeti Sangwan. Path finding: A* or dijkstras. International Journal in IT and Engineering, 2(01), 2014.
13. Mengzhe Zhang. Path planning for autonomous vehicles. PhD thesis, 2014.
14. B Moses Sathyaraj, Lakhmi C Jain, Anthony Finn, and S Drake. Multiple uavs path planning algorithms: a comparative study. Fuzzy Optimization and Decision Making, 7(3):257, 2008.
15. K. Khantanapoka and K. Chinnasarn. Pathfinding of 2d 3d game real-time strategy with depth direction a algorithm for multi-layer. In 2009 Eighth International Symposium on Natural Language Processing, pages 184-188, Oct 2009.
16. Andris Jansons. A star (a*) path finding c, May 2018.
17. Making your own linear filters, 2017.