Universidad de La Laguna

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO DE FIN DE GRADO

SETIEMBRE 2016

Algoritmos de Mapeo para Robótica Móvil y Entorno de Simulación

Diego Alberto Tamayo Guzmán

Tutores:

Leopoldo Acosta Sánchez
Antonio Luis Morell Gonzáles

Resumen

El presente documento versa sus contenidos sobre el desarrollo y evaluación de diferentes algoritmos computacionales orientados al problema de robótica móvil conocido como *mapeo*, que consiste en crear un mapa del entorno en el que se encuentre el robot usando la información procedente de los sensores durante la exploración del entorno. La complejidad de dicho problema va en aumento a medida que implementamos nuevas características como detección de errores, localización por estimación, mapas probabilísticos, etc.

Así, conforme avanza el proyecto, se van proponiendo nuevas mejoras e implementando algoritmos cada vez más complejos. Con el objetivo de evaluar el comportamiento, eficacia y resultados de cada algoritmo, se desarrollan diferentes entornos de simulación adaptados a cada uno de estos. Todos los resultados de las simulaciones se presentan de forma gráfica y concisa, realizando los análisis pertinentes en cada caso.

Los problemas presentados en los algoritmos se corrigen en la siguiente versión junto con nuevas características añadidas, de forma que la versión final implementa todas las características vistas y presenta buenos resultados, cumpliendo así los objetivos propuestos.

Abstract

This document focuses its contents on the development and evaluation of different computational algorithms with the primary aim of solving the mobile robotics problem known as *mapping*, this problem is that of acquiring a spatial model of a robot's environment by analyzing and construing data achieved by the sensors during an environment exploration. The problem's complexity increases as we implement new features such as error detection, estimation-based localization, probabilistic maps, etc.

As the project progresses, further improvements are proposed and implemented in increasingly complex algorithms. In order to evaluate the behavior, performance and results of each algorithm, simulation environments have been developed adapted to each of these. All simulation results are presented graphically and concisely, making relevant analysis when needed.

Issues found on the algorithms have been corrected in following versions adding new features as well, so that the final version implements all the features seen and achieves good results, fulfilling the project goals.

Índice

1. Introducción **19**

1.1. Definición de Robot 19

1.2. El Problema de la Navegación 20

1.3. Mapeo 21

1.4. Caso particular 22

1.4.1. Hardware 22

1.4.2. Ruido/Errores 23

1.4.3. Software 23

1.5. Objetivos del Proyecto 24

2. Primeras Versiones **25**

2.1. Interpretar los datos del sensor, crear un mapa simple 25

2.1.1. Cambio de coordenadas 25

2.1.2. Cambio de sistema de referencia 26

2.1.3. Implementación 27

2.2. Simulador del telémetro 27

2.2.1. Definir las características del entorno 28

2.2.2. Referenciar los obstáculos al sistema de referencia del robot 28

2.2.3. Encontrar los impactos 29

2.3. Entorno de Simulación 30

2.3.1. Segunda versión 31

2.4. Cinemática del robot 32

2.4.1. Implementación 34

2.4.2. Entorno de Simulación 34

3. Introduciendo ruidos **35**

3.1. Implementando el ruido en el simulador 36

3.1.1. Entorno de Simulación 36

3.2. Corrigiendo el error de odometría 37

3.2.1. *Occupancy Grid* . . . 38
3.2.1.1. Implementación . . . 38
3.2.1.2. Entorno de Simulación . . . 39
3.2.2. *Map Matching* . . . 40
3.2.2.1. Implementación . . . 43
3.2.2.2. Entorno de Simulación . . . 44
3.2.2.3. Evaluación de Resultados . . . 44
3.3. Corrigiendo el error de medida . . . 45
3.3.1. Implementación . . . 46
3.3.2. Entorno de Simulación . . . 47
3.3.3. Evaluación de Resultados . . . 47

4. Versiones Finales **49**
4.1. Versión 7 del Algoritmo de Mapeo . . . 49
4.1.1. Implementación . . . 50
4.1.2. Entorno de Simulación . . . 50
4.1.3. Evaluación de Resultados . . . 50
4.2. Versión 8 del Algoritmo de Mapeo . . . 51
4.2.1. Implementación . . . 52
4.2.2. Entorno de Simulación . . . 52
4.2.3. Evaluación de Resultados . . . 52
4.3. Versión 9 del Algoritmo de Mapeo . . . 53
4.3.1. Implementación . . . 54
4.3.2. Entorno de Simulación . . . 54
4.3.3. Evaluación de Resultados . . . 55

Conclusions **57**

Apéndices **59**

A. Resultados de las Simulaciones **61**
A.1. MappingTB_v1 . . . 62
A.2. MappingTB_v2_1 . . . 63
A.3. MappingTB_v2_2 . . . 64
A.4. MappingTB_v3_1 . . . 65
A.5. MappingTB_v4_1 . . . 66
A.6. MappingTB_v5_1 . . . 71

A.7. MappingTB_v5_2 . . . 74
A.8. MappingTB_v6_1 . . . 75
A.9. MappingTB_v6_2 . . . 87
A.10. MappingTB_v6_3 . . . 99
A.11. MappingTB_v7_1 . . . 101
A.12. MappingTB_v7_2 . . . 105
A.13. MappingTB_v7_3 . . . 111
A.14. MappingTB_v8_1 . . . 115
A.15. MappingTB_v8_2 . . . 128
A.16. MappingTB_v9_1 . . . 136
A.17. MappingTB_v9_2 . . . 145
A.18. MappingTB_v10_1 . . . 153
A.19. MappingTB_v10_2 . . . 171
A.20. MappingTB_v10_3 . . . 179

B. Códigos **189**
B.1. Mapping . . . 189
B.1.1. Mapping_v1 . . . 189
B.1.2. Mapping_v2 . . . 190
B.1.3. Mapping_v3 . . . 191
B.1.4. Mapping_v4 . . . 192
B.1.5. Mapping_v5 . . . 192
B.1.6. Mapping_v6 . . . 194
B.1.7. Mapping_v7 . . . 195
B.1.8. Mapping_v8 . . . 197
B.1.9. Mapping_v9 . . . 199
B.2. VissionSimulation . . . 201
B.2.1. VissionSimulation_v1 . . . 201
B.3. MappingTB . . . 202
B.3.1. MappingTB_v1 . . . 202
B.3.2. MappingTB_v2 . . . 203
B.3.2.1. MappingTB_v2_1 . . . 203
B.3.2.2. MappingTB_v2_2 . . . 204
B.3.3. MappingTB_v3 . . . 205
B.3.4. MappingTB_v4 . . . 206
B.3.5. MappingTB_v5 . . . 207

B.3.5.1. MappingTB_v5_1 . . . 207
B.3.5.2. MappingTB_v5_2 . . . 209
B.3.6. MappingTB_v6 . . . 210
B.3.6.1. MappingTB_v6_1 . . . 210
B.3.6.2. MappingTB_v6_2 . . . 212
B.3.6.3. MappingTB_v6_3 . . . 213
B.3.7. MappingTB_v7 . . . 215
B.3.7.1. MappingTB_v7_1 . . . 215
B.3.7.2. MappingTB_v7_2 . . . 216
B.3.7.3. MappingTB_v7_3 . . . 217
B.3.8. MappingTB_v8 . . . 219
B.3.8.1. MappingTB_v8_1 . . . 219
B.3.8.2. MappingTB_v8_2 . . . 221
B.3.9. MappingTB_v9 . . . 222
B.3.9.1. MappingTB_v9_1 . . . 222
B.3.9.2. MappingTB_v9_2 . . . 224
B.3.10. MappingTB_v10 . . . 226
B.3.10.1. MappingTB_v10_1 . . . 226
B.3.10.2. MappingTB_v10_2 . . . 227
B.3.10.3. MappingTB_v10_3 . . . 229

C. Funciones Complementarias **233**
C.1. dibujar_v1 . . . 233
C.2. dibujar_v2 . . . 236
C.3. print_table . . . 238
C.4. CalculateMAP_v1 . . . 239
C.5. DisplayMap . . . 242
C.6. CalculateMAP_v2 . . . 244
C.7. CalculateMAP_v3 . . . 247

D. Pruebas Complementarias **251**
D.1. Mapping_v5: Bucle doble VS media . . . 251

Bibliografía **253**

Índice de figuras

1.1. Subproblemas de la navegación 20
1.2. Encoder óptico montado en motor 23
1.3. Telémetro láser Sick LMS-100 23

2.1. Cambio de coordenadas 25
2.2. Cambio de sistema de referencia 26
2.3. Ejemplo de entorno 28
2.4. Simulación de los impactos 29
2.5. Diagrama de *MappingTB_v1* 30
2.6. Diagrama de *MappingTB_v2* 31
2.7. Modelo de locomoción diferencial 32
2.8. Dimensiones del robot 33
2.9. Diagrama de *MappingTB_v3* 34

3.1. Modelo de error gaussiano 35
3.2. Diagrama de *MappingTB_v4* 37
3.3. Diagrama de *MappingTB_v5* 40
3.4. Ventana de comparación de mapas 41
3.5. Mapa de *MappingTB_v5_2* sin error de distancia 45
3.6. Mapa de *MappingTB_v5_2* con error de distancia 0.2 45
3.7. Ampliación del resultado de *MappingTB_v6_2* con error de distancia 0.2 . . 48
3.8. Ampliación del resultado de *MappingTB_v7_2* con error de distancia 0.2 . . 48
3.9. Mapa resultante de *MappingTB_v6_3(0,0)* 48
3.10. Mapa resultante de *MappingTB_v7_3(0,0)* 48

4.1. Diagrama de la *comparación híbrida por máscara* 49

A.1. Resultados de *MappingTB_v1* 62
A.2. Resultados de *MappingTB_v2_1* 63
A.3. Resultados de *MappingTB_v2_2* 64

A.4. Resultados de *MappingTB_v3_1* 65
A.5. Resultado gráfico de *MappingTB_v4* con error de medida 66
A.6. Resultado gráfico de *MappingTB_v4* con error de odometría 67
A.7. Gráfica del error de pose en *MappingTB_v4_1*(0.8,0) 70
A.8. Resultado gráfico de *MappingTB_v5_1 sin errores* 71
A.9. Resultado gráfico de *MappingTB_v5_1 con error de odometría* 72
A.10. Resultado gráfico de *MappingTB_v5_1 con error de medida* 73
A.11. Resultado gráfico de *MappingTB_v5_2 con error de odometría* 74
A.12. Resultado gráfico de *MappingTB_v6_1* con error de odometría 0,5 75
A.13. Mapa resultante de *MappingTB_v6_1* con error de odometría 0,5 76
A.14. Gráfica del error de pose en *MappingTB_v6_1*(0.5,0) 78
A.15. Resultado gráfico de *MappingTB_v6_1* con error de odometría 0,8 79
A.16. Mapa resultante de *MappingTB_v6_1* con error de odometría 0,8 80
A.17. Gráfica del error de pose en *MappingTB_v6_1*(0.8,0) 82
A.18. Resultado gráfico de *MappingTB_v6_1* con error de odometría 1,1 83
A.19. Mapa resultante de *MappingTB_v6_1* con error de odometría 1,1 84
A.20. Gráfica del error de pose en *MappingTB_v6_1*(1.1,0) 86
A.21. Resultado gráfico de *MappingTB_v6_2* con error de odometría 0,16 87
A.22. Mapa resultante de *MappingTB_v6_2* con error de odometría 0,16 88
A.23. Gráfica del error de pose en *MappingTB_v6_2*(0.16,0) 90
A.24. Resultado gráfico de *MappingTB_v6_2* con error de odometría 0,3 91
A.25. Mapa resultante de *MappingTB_v6_2* con error de odometría 0,3 92
A.26. Gráfica del error de pose en *MappingTB_v6_2*(0.3,0) 94
A.27. Resultado gráfico de *MappingTB_v6_2* con error de odometría 0,3 y de distancia 0,1 95
A.28. Mapa resultante de *MappingTB_v6_2* con error de odometría 0,3 y de distancia 0,1 96
A.29. Gráfica del error de pose en *MappingTB_v6_2*(0.3,0.1) 98
A.30. Resultado gráfico de *MappingTB_v6_3* sin error 99
A.31. Mapa resultante de *MappingTB_v6_3(0,0)* 100
A.32. Resultado gráfico de *MappingTB_v7_1* sin error 101
A.33. Ampliación del resultado final de *MappingTB_v7_1* sin error 102
A.34. Mapa resultante de *MappingTB_v7_1(0,0)* 102
A.35. Resultado gráfico de *MappingTB_v7_1* con error de distancia 0.1 103
A.36. Ampliación del resultado final de *MappingTB_v7_1* con error de distancia 0.1 104
A.37. Mapa resultante de *MappingTB_v7_1(0,0.1)* 104

A.38. Resultado gráfico de *MappingTB_v7_2* con error de distancia 0.1 105
A.39. Ampliación del resultado final de *MappingTB_v7_2* con error de distancia 0.1 106
A.40. Mapa resultante de *MappingTB_v7_2(0,0.1)* 106
A.41. Resultado gráfico de *MappingTB_v7_2* con error de distancia 0.2 107
A.42. Ampliación del resultado final de *MappingTB_v7_2* con error de distancia 0.2 108
A.43. Mapa resultante de *MappingTB_v7_2(0,0.2)* 108
A.44. Resultado gráfico de *MappingTB_v7_2* con error de odometría 0.3 109
A.45. Mapa resultante de *MappingTB_v7_2(0.3,0)* 110
A.46. Resultado gráfico de *MappingTB_v7_3* sin error 111
A.47. Ampliación del resultado final de *MappingTB_v7_3* sin error 112
A.48. Mapa resultante de *MappingTB_v7_3(0,0)* 112
A.49. Resultado gráfico de *MappingTB_v7_3* con error de distancia 0.1 113
A.50. Ampliación del resultado final de *MappingTB_v7_3* con error de distancia 0.1 114
A.51. Mapa resultante de *MappingTB_v7_3 (0,0.1)* 114
A.52. Resultado gráfico de *MappingTB_v8_1* con error de odometría 0.5 115
A.53. Mapa resultante de *MappingTB_v8_1(0.5,0)* 116
A.54. Gráfica del error de pose en *MappingTB_v8_1*(0.5,0) 118
A.55. Resultado gráfico de *MappingTB_v8_1* con error de odometría 0.8 119
A.56. Mapa resultante de *MappingTB_v8_1(0.8,0)* 120
A.57. Gráfica del error de pose en *MappingTB_v8_1*(0.8,0) 122
A.58. Desempeño temporal de *MappingTB_v8_1(0.8,0)* 123
A.59. Gráfica de errores cuadráticos medios para múltiples simulaciones de *MappingTB_v8_1(0.8,0)* 123
A.60. Resultado gráfico de *MappingTB_v8_1* con error de odometría 1.1 124
A.61. Mapa resultante de *MappingTB_v8_1(1.1,0)* 125
A.62. Gráfica del error de pose en *MappingTB_v8_1*(1.1,0) 127
A.63. Resultado gráfico de *MappingTB_v8_2* con error de odometría 0.16 128
A.64. Mapa resultante de *MappingTB_v8_1(0.16,0)* 129
A.65. Gráfica del error de pose en *MappingTB_v8_2*(0.16,0) 131
A.66. Resultado gráfico de *MappingTB_v8_2* con error de odometría 0.3 132
A.67. Mapa resultante de *MappingTB_v8_1(0.3,0)* 133
A.68. Gráfica del error de pose en *MappingTB_v8_2*(0.3,0) 135
A.69. Resultado gráfico de *MappingTB_v9_1* con error de odometría 0.5 136
A.70. Mapa resultante de *MappingTB_v9_1(0.5,0)* 137
A.71. Gráfica del error de pose en *MappingTB_v9_1*(0.5,0) 139
A.72. Resultado gráfico de *MappingTB_v9_1* con error de odometría 0.8 140

A.73. Mapa resultante de *MappingTB_v1_2(0.8,0)* 141
A.74. Gráfica del error de pose en *MappingTB_v9_1*(0.8,0) 143
A.75. Desempeño temporal de *MappingTB_v9_1(0.8,0)* con precisión 1 144
A.76. Resultado gráfico de *MappingTB_v9_2* con error de odometría 0.16 145
A.77. Mapa resultante de *MappingTB_v9_2(0.16,0)* 146
A.78. Gráfica del error de pose en *MappingTB_v9_2*(0.16,0) 148
A.79. Resultado gráfico de *MappingTB_v9_2* con error de odometría 0.3 149
A.80. Mapa resultante de *MappingTB_v9_2(0.3,0)* 150
A.81. Gráfica del error de pose en *MappingTB_v9_2*(0.3,0) 152
A.82. Resultado gráfico de *MappingTB_v10_1* con error de odometría 0.5 153
A.83. Mapa resultante de *MappingTB_v10_1(0.5,0)* 154
A.84. Gráfica del error de pose en *MappingTB_v10_1*(0.5,0) 156
A.85. Resultado gráfico de *MappingTB_v10_1* con errores de odometría 0.5 y de distancia 0.1 157
A.86. Ampliación del resultado final de *MappingTB_v10_1* con errores de odometría 0.5 y de distancia 0.1 158
A.87. Mapa resultante de *MappingTB_v10_1(0.5,0.1)* 158
A.88. Gráfica del error de pose en *MappingTB_v10_1*(0.5,0.1) 161
A.89. Resultado gráfico de *MappingTB_v10_1* con error de odometría 0.8 162
A.90. Mapa resultante de *MappingTB_v10_1(0.8,0)* 163
A.91. Gráfica del error de pose en *MappingTB_v10_1*(0.8,0) 165
A.92. Resultado gráfico de *MappingTB_v10_1* con errores de odometría 0.8 y de distancia 0.1 166
A.93. Ampliación del resultado final de *MappingTB_v10_1* con errores de odometría 0.8 y de distancia 0.1 167
A.94. Mapa resultante de *MappingTB_v10_1(0.8,0.1)* 167
A.95. Gráfica del error de pose en *MappingTB_v10_1*(0.8,0.1) 170
A.96. Resultado gráfico de *MappingTB_v10_2* con errores de odometría 0.16 y de distancia 0.05 171
A.97. Ampliación del resultado final de *MappingTB_v10_2* con errores de odometría 0.16 y de distancia 0.05 172
A.98. Mapa resultante de *MappingTB_v10_2(0.16,0.05)* 172
A.99. Gráfica del error de pose en *MappingTB_v10_2*(0.16,0.05) 174
A.100. Resultado gráfico de *MappingTB_v10_2* con errores de odometría 0.3 y de distancia 0.1 175
A.101. Ampliación del resultado final de *MappingTB_v10_2* con errores de odometría 0.3 y de distancia 0.1 176
A.102. Mapa resultante de *MappingTB_v10_2(0.3,0.1)* 176

A.103. Gráfica del error de pose en *MappingTB_v10_2*(0.3,0.1) 178
A.104. Resultado gráfico de *MappingTB_v10_3* con errores de odometría 0.16 y de distancia 0.05 179
A.105. Ampliación del resultado final de *MappingTB_v10_3* con errores de odometría 0.16 y de distancia 0.05 180
A.106. Mapa resultante de *MappingTB_v10_3(0.16,0.05)* 180
A.107. Gráfica del error de pose en *MappingTB_v10_3*(0.16,0.05) 183
A.108. Desempeño temporal de *MappingTB_v10_3(0.16,0.05)* 183
A.109. Resultado gráfico de *MappingTB_v10_3* con errores de odometría 0.2 y de distancia 0.1 184
A.110. Ampliación del resultado final de *MappingTB_v10_3* con errores de odometría 0.2 y de distancia 0.1 185
A.111. Mapa resultante de *MappingTB_v10_3(0.2,0.1)* 185
A.112. Gráfica del error de pose en *MappingTB_v10_3*(0.2,0.1) 188
A.113. Gráfica de errores cuadráticos medios para múltiples simulaciones de *MappingTB_v10_3(0.16,0.05)* 188

C.1. Ejemplo de funcionamiento de *dibujar_v1* 233
C.2. Ejemplo 1 de funcionamiento de *dibujar_v2* 236
C.3. Ejemplo 2 de funcionamiento de *dibujar_v2* 236
C.4. Occupancy Grid 239
C.5. Resultado de *DisplayMap* con mapa booleano 242
C.6. Resultado de *DisplayMap* con mapa probabilístico 243
C.7. Metodo geométrico de *CalculateMAP_v2* 244
C.8. Desempeño temporal de *CalculateMAP_v1* 245
C.9. Desempeño temporal de *CalculateMAP_v1* 245
C.10. Diagrama de los incrementos y decrementos para la *Occupancy Grid probabilística* 247

D.1. Desempeño temporal de *Mapping_v5* con bucle de precisión 1 251
D.2. Desempeño temporal de *Mapping_v5* con bucle de precisión 2 252
D.3. Desempeño temporal de *Mapping_v5* con media de AND 252

Índice de tablas

1.1. Desviaciónes típicas consideradas para los errores 23

A.1. Error de pose en *MappingTB_v4*(0.8,0) 69
A.2. Error de pose en *MappingTB_v6_1*(0.5,0) 78
A.3. Error de pose en *MappingTB_v6_1*(0.8,0) 82
A.4. Error de pose en *MappingTB_v6_1*(1.1,0) 86
A.5. Error de pose en *MappingTB_v6_2*(0.16,0) 90
A.6. Error de pose en *MappingTB_v6_2*(0.3,0) 94
A.7. Error de pose en *MappingTB_v6_2*(0.3,0.1) 98
A.8. Error de pose en *MappingTB_v8_1*(0.5,0) 118
A.9. Error de pose en *MappingTB_v8_1*(0.8,0) 122
A.10. Error de pose en *MappingTB_v8_1*(1.1,0) 127
A.11. Error de pose en *MappingTB_v8_2*(0.16,0) 131
A.12. Error de pose en *MappingTB_v8_2*(0.3,0) 135
A.13. Error de pose en *MappingTB_v9_1*(0.5,0) 139
A.14. Error de pose en *MappingTB_v9_1*(0.8,0) 143
A.15. Error de pose en *MappingTB_v9_2*(0.16,0) 148
A.16. Error de pose en *MappingTB_v9_2*(0.3,0) 152
A.17. Error de pose en *MappingTB_v10_1*(0.5,0) 156
A.18. Error de pose en *MappingTB_v10_1*(0.5,0.1) 160
A.19. Error de pose en *MappingTB_v10_1*(0.8,0) 165
A.20. Error de pose en *MappingTB_v10_1*(0.8,0.1) 169
A.21. Error de pose en *MappingTB_v10_2*(0.16,0.05) 174
A.22. Error de pose en *MappingTB_v10_2*(0.3,0.1) 178
A.23. Error de pose en *MappingTB_v10_3*(0.16,0.05) 182
A.24. Error de pose en *MappingTB_v10_3*(0.2,0.1) 187

Capítulo 1

Introducción

1.1. Definición de Robot

Dar una definición concreta de robot no es sencillo, ya que es un concepto tan amplio y que abarca tantos aspectos que requiere un gran nivel de abstracción. Se puede definir como una entidad física hecha por el hombre con el propósito de reemplazarlo a este en una o varias tareas que posee una cierta conexión de retroalimentación inteligente entre el sentido y la acción para cumplir su propósito.

La RIA(*Robotics Industries Asociation*) define al robot como cualquier estructura mecánica que opera con un cierto grado de autonomía, bajo el control de un computador, para la realización de una tarea, y que dispone de un sistema sensorial más o menos evolucionado para obtener información de su entorno.

Nosotros definiremos al robot como un dispositivo generalmente electro-mecánico que desempeña tareas con un cierto grado de autonomía siguiendo un conjunto de reglas predefinidas, suponiendo un cierto de grado de inteligencia artificial. Sus componentes se clasifican en tres grandes grupos que responden al sentido, la inteligencia y la acción. Según esta definición, podemos decir que en la vida cotidiana estamos rodeados de robots, como son el horno microondas, la lavadora, los sistemas autónomos del coche, etc.

Entonces, definimos las tres partes básicas de un robot:

Sensores Son todos aquellos dispositivos que permiten al robot obtener información del entorno como si de sentidos se tratase. Dicha información puede ser de diversa naturaleza, ya sea visual, auditiva, electromagética, atmoesférica, etc. Ejm: cámaras, botones, micrófonos, sensores varios, etc.

Actuadores Son todos los dispositivos que permiten al robot interactuar con el entorno, es decir, realizar acciones físicas que afecten a sí mismo o a su entorno. Ejm: motores, altavoces, pantallas, luces, etc.

Controlador Es el dispositivo o conjunto de dispositivos que se encarga de recibir la información de los sensores, procesarla y en base a esta y a su programación interna, enviar ordenes a los actuadores.

Luego está el grupo de los robots móviles, que son aquellos robots que no están fijos a una posición sino que pueden moverse con un cierto grado de libertad por su entorno. Se basan en

carros o plataformas dotados de un sistema locomotor normalmente rodante y suelen contar con un nivel relativamente elevado de inteligencia. Entre los robots móviles que podemos observar en la vida cotidiana están los robots de limpieza, los helicópteros y cuadracópteros teledirigidos, robots cortacésped, robots de LEGO, etc. Es este grupo al que se dedica el presente estudio.

1.2. El Problema de la Navegación

La navegación es una rama fundamental de la robótica móvil, presente en casi todos los robots con cierto nivel de complejidad. Se define esta como la capacidad de dado un punto de partida A, alcanzar los puntos de destino B utilizando su conocimiento y la información sensorial que recibe.

De esta forma, todo robot móvil que pretenda cambiar de posición con cierto grado de autonomía necesita contar con un sistema de navegación más o menos complejo dependiendo de aspectos como la complejidad del sistema locomotor, los grados de libertad en el movimiento o la complejidad del entorno.

La navegación implica resolver los siguientes subproblemas:

Percepción Interpretar los datos que le suministran sus sensores para extraer información útil.

Localización Determinar su posición en el entorno.

Planificación Decidir como actuar para alcanzar el objetivo.

Control de Movimiento Gestionar sus actuadores para conseguir la trayectoria deseada.

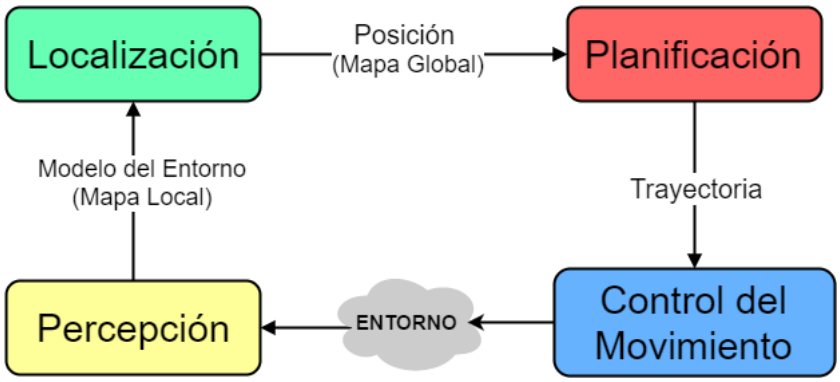


Figura 1.1: Subproblemas de la navegación

La localización se define como la capacidad del robot de conocer su posición actual con referencia a un sistema de coordenadas y es el pilar principal de la navegación. Generalmente, un robot se localiza utilizando tanto un conocimiento previo que tenga del entorno como la información sensorial que pueda recibir. Ese conocimiento previo del entorno es llamado mapa y es en este que se centra el trabajo de este documento, concretamente en su adquisición, ya sea en un paso previo a la navegación o como parte incluida en la localización.

1.3. Mapeo

Se conoce a mapeo como el proceso de obtención/generación del mapa del entorno necesario para la navegación de robots móviles.

Existen diferentes métodos de mapeo, definiéndose el *SLAM(Simultaneous Localization and Mapping)* como el método usado cuando generamos el mapa mediante una exploración del entorno. Como su nombre lo dice, requiere de realizar en paralelo al proceso de mapeo, un proceso de localización ya que la posición es una variable necesaria a la hora de generar el mapa con este método.

La solución a este problema ha sido objeto de estudio por parte de la comunidad científica durante los últimos 20 años. El ruido presente en sistemas sensoriales, los inevitables errores y aproximaciones cometidos en los modelos empleados y la dificultad representativa de los entornos a medida que éstos aumentan en complejidad, hace que la tarea de resolver el mencionado problema sea ardua. Tal complejidad tiene una doble vertiente:

- Desde un punto de vista conceptual, se busca la manera de obtener y manipular datos sobre el entorno haciendo de uso de sensores que distan mucho de ser perfectos. Surgen preguntas como ¿Qué es el entorno? ¿Qué geometrías cabe esperar en él? ¿Existen objetos móviles? ¿Con qué precisión es necesario representarlo?

- Desde un punto de vista computacional, el modo en que el robot percibe su entorno, la cantidad de información disponible así como las técnicas empleadas en su procesamiento, interpretación y combinación, determinará los recursos computacionales necesarios para la construcción del mapa. Estos recursos no son ilimitados, menos aun si el objetivo es ceñirse a los disponibles a bordo de la máquina. Por ello los interrogantes en este caso se refieren a la eficiencia del algoritmo y la posibilidad de que su implementación sea posible en tiempo real.

Los mapas generados pueden ser de dos tipos:

De representación continua Elementos representados en un espacio de coordenadas continuo.

De representación troceada Descomposición del entorno aproximada por celdas de tamaño fijo o adaptativo, cuyos valores pueden ser de tipo booleano o probabilístico.

La localización se suele hacer mediante estimación o *dead reckoning*, procedimiento matemático que utiliza fórmulas trigonométricas para determinar la ubicación actual basándose en el rumbo y la velocidad de navegación a lo largo de un periodo. La implementación más simple posible de esta técnica se conoce como *odometría*, que emplea sensores para determinar la velocidad de las ruedas de un robot y mediante un modelo físico-matemático[1], deducir el cambio de posición en función de estas en un intervalo de tiempo.

En algunos casos, el SLAM incluye también un proceso de estimación de trayectorias paralelo a los dos procesos mencionados, pero no será ese nuestro caso, ya que utilizaremos trayectorias prefijadas.

Mas adelante se profundizará en cada uno de los aspectos mencionados mientras se avanza en complejidad y se van implementando gradualmente.

[1] Dicho modelo dependerá de la configuración del sistema locomotor del robot. Los más habituales son el sistema de locomoción diferencial y el sistema de triciclo.

1.4. Caso particular

Para el desarrollo de este proyecto, definiremos ciertos aspectos referentes al punto de partida, como el hardware en el que nos basaremos, los errrores a considerar o el software que se utilizará.

1.4.1. Hardware

Se considerará un robot móvil simple con un sistema de locomoción diferencial del que se hablará más adelante. Los elementos de este serán:

- **Actuadores**

  Motores del sistema de locomoción diferencial Tal como se explica más adelante, el sistema de locomoción diferencial cuenta con dos motores de los cuales dependerá el movimiento del robot. Nótese que al usar trayectorias prefijadas, no vamos realmente a mandar información a los actuadores sino que asumiremos que estos se mueven debido a un tercer sistema de control.

- **Sensores**

  Sensores de velocidad angular en las ruedas Se considerará un sensor de velocidad angular por cada rueda, dicho sensor puede ser un encoder ópticos u otro similar que responda al mismo propósito. Nótese que tampoco se tiene en cuenta el algoritmo de control de los mismos sino que se asume las velocidades como muestreadas en el tiempo y conocido su valor en cada periodo de muestreo.

  Telémetro láser Un telémetro es un dispositivo que utiliza un rayo láser para determinar la distancia hasta un objeto en base al tiempo de vuelo de la luz. Concretamente, el que se suele usar para estos casos es capaz de realizar un barrio de frecuencias en un cierto rango (que consideraremos 270 grados sexagesimales de apertura) y con un cierto incremento (que consideraremos de medio grado sexagesimal). Es decir, que realiza un cierto número de mediciones para cada ángulo del barrido y nos comunica las distancias encontradas para cada uno de estos. A estas distancias individuales se les denomina *impactos* y pueden ser positivos (de encontrarse un obstáculo) o nulos (de no encontrar obstáculo para una cierta longitud máxima propia del telémetro. De forma análoga a los sensores de velocidad angular, no se ha tenido en cuenta el control de dicho sensor sino que se tiene como conocido el conjunto de distancias para cada medición.

- **Controlador.-** Se considera que el robot tendrá un PC montado en la carrocería, por lo que sus características computacionales serán equivalentes al PC usado para las simulaciones. Características del equipo de simulación:

  Procesador Intel core i5 2500K

  Memoria RAM 8 Gb DDR3 1600MHz

  Disco Duro HDD

  Tarjeta gráfica Nvidia GeForce GTX960 4Gb

  Sistema Operativo Windows 10, 64 bits

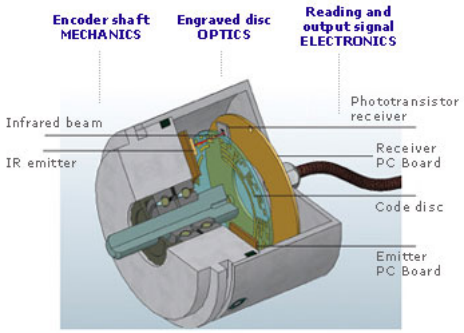


Figura 1.2: Encoder óptico montado en motor

Figura 1.3: Telémetro láser Sick LMS-100

1.4.2. Ruido/Errores

Respecto al ruido, se tendrán en cuenta dos errores, ambos de tipo gaussiano. Es decir, que responden a una distribución normal de media 0 y desviación típica variable.

Error de Odometría Responde a la imperfección en la medida de las velocidades angulares de cada rueda debida al sensor mismo o al canal de transmisión de datos.

Error de distancia/medida Responde a la imperfección en la medida de las distancias o *impactos* del telémetro láser debida al sensor mismo o al canal de transmisión de datos.

Error	rango de medidas	Desv. típica en condiciones normales	Otras desv. típicas consideradas
odometría	$0 - 60 rad/s$	$0,5 rad/s$	$0,8 rad/s / 0,1 rad/s$
	$0 - 20 rad/s$	$0,16 rad/s$	$0,3 rad/s$
distancia	$0 - 30 m$	$0,05 m$	$0,1 m$

Tabla 1.1: Desviaciónes típicas consideradas para los errores

1.4.3. Software

Todos los algoritmos se han desarrollado usando el programa *Matlab* de la empresa *Mathworks*, ya que es un software de gran potencia que cuenta con muchas herramientas útiles para servir a nuestros propósitos. La versión utilizada es la *R2016a*.

1.5. Objetivos del Proyecto

El Objetivo principal de este proyecto es desarrollar un algoritmo de mapeo tipo SLAM que trabaje adecuadamente bajo las condiciones de ruidos descritas. El mapa final debe ser representativo de la realidad y tener un formato adecuado para poder usarse posteriormente en navegación. Precisamente por esto, se ha acordado que los obstáculos móviles no deberán mostrarse en el mapa, ya que para una correcta localización, se debe usar un mapa de obstáculos fijos. Además, el algoritmo debe presentar la sencillez/eficiencia necesarias para que su tiempo de ejecución sea lo suficientemente bajo y permita su ejecución en tiempo real.

Objetivos secundarios:

- Desarrollar un entorno de simulación adecuado que permita probar los distintos algoritmos de mapeo que se vayan desarrollando y permita visualizar los resultados de forma simple y concisa.
- Poder exportar los resultados de las simulaciones en distintos formatos para su posterior análisis.
- Desarrollar algoritmos de alta calidad que respondan a demandas de robustez, universalidad, parametización, customización, fácil modificación, fácil traducción a otro lenguaje/-plataforma, etc.
- Presentar el trabajo realizado de forma ordenada y con todos los resultados de las simulaciones necesarios para su correcta exposición y evaluación.
- Mejorar nuestro nivel de expertise en áreas como la robótica, localización, programación, entornos de simulación, etc.

Capítulo 2

Primeras Versiones

2.1. Interpretar los datos del sensor, crear un mapa simple

El sensor con el que contamos es un telémetro láser, que realiza un barrido de ángulos y nos da las distancias de los impactos con el objeto más cercano en cada uno de dichos ángulos. Para poder trabajar con estos datos se deben realizar una serie de operaciones.

2.1.1. Cambio de coordenadas

Ya que tenemos un conjunto de distancias para diferentes ángulos, se puede decir que estos datos se corresponden a coordenadas polares respecto a la posición del robot, por lo tanto, habrá que realizar un cambio a coordenadas cartesianas donde:

$$x = \rho \cdot cos(\theta) \tag{2.1}$$

$$y = \rho \cdot sen(\theta) \tag{2.2}$$

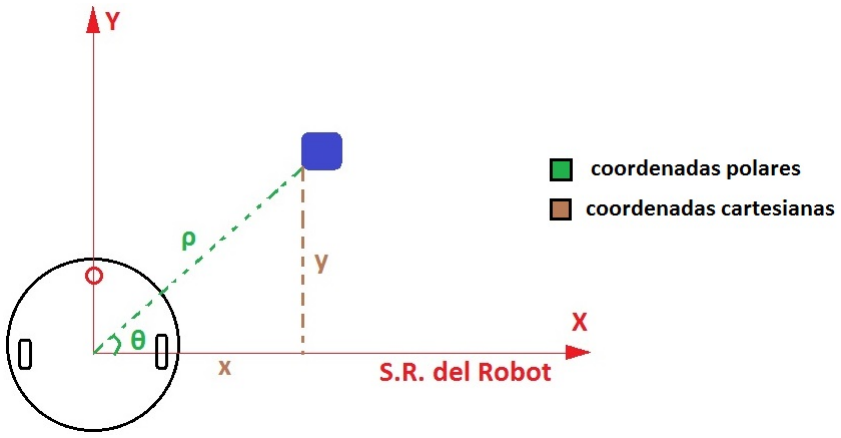


Figura 2.1: Cambio de coordenadas

2.1.2. Cambio de sistema de referencia

Ahora debemos cambiar el sistema de referencia de todas las coordenadas cartesianas obtenidas, ya que estas se encuentran referenciadas al centro del robot (posición del telémetro) pero para poder trabajar con un mapa global, debemos referenciarlas a un sistema externo e inmóvil, al que a partir de ahora llamaremos *Sistema de Referencia Global*.

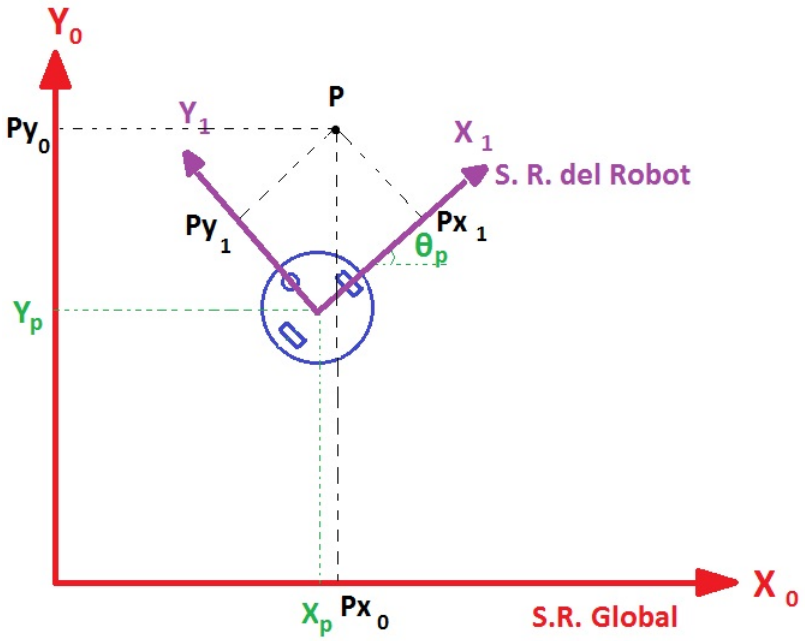


Figura 2.2: Cambio de sistema de referencia

Como podemos apreciar, es necesario definir y conocer tres nuevas variables que indiquen la posición del robot respecto al sistema de referencia global. Al conjunto de estas tres variables se le llamará a partir de ahora *pose*.

$$Pose = \begin{bmatrix} x_p \\ y_p \\ \theta p \end{bmatrix} \qquad P_1 = \begin{bmatrix} Px_1 \\ Py_1 \\ 1 \end{bmatrix} \qquad P_0 = \begin{bmatrix} Px_0 \\ Py_0 \\ 1 \end{bmatrix}$$

Para realizar el cambio de sistema de referencia, tendremos que definir una matriz ${}^0T_1$ tal que:

$${}^0T_1 \cdot P_1 = P_0 \tag{2.3}$$

$${}^0T_1 = \begin{bmatrix} cos(\theta) & -sen(\theta) & x_p \\ sen(\theta) & cos(\theta) & y_p \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

2.1.3. Implementación

Para poder implementar lo visto en un algoritmo computacional, debemos primero definir ciertos aspectos clave:

- El telémetro nos proporcionará un vector de distancias correspondientes al barrido de ángulos, donde vamos a asumir que:
  - El barrido se realiza en el sentido de las agujas del reloj, es decir que para mi sistema de referencia, se empieza en el ángulo mayor y se termina en el menor.
  - El telémetro cuenta con una apertura de visión de 270 grados sexagesimales, es decir que para mi sistema de referencia, el primer ángulo del barrido será 225º y el último -45º.
  - La resolución del telémetro es de 0.5 grados sexagesimales, es decir, que la distancia entre un ángulo del barrido y el siguiente será de medio grado, por lo que tendremos 541 distancias por medición.
- El producto final será un conjunto de coordenadas correspondientes a los impactos del telémetro y referenciadas al sistema de referencia global.

Además, intentaremos estructurar el algoritmo de forma que sea robusto y paramétrico.[1]

La función sido llamada Mapping_v1 por ser la primera versión del algoritmo de mapeo. Dicha función recibe como argumentos de entrada la pose del robot(*x,y,theta*) y el vector de distancias *RAD*[2]); y devuelve dos vectores de coordenadas cartesianas correspondientes a las componente X e Y de los impactos del telémetro, considerados como la forma más primitiva de mapa. El código se puede consultar en el apéndice B.

2.2. Simulador del telémetro

Antes de proceder a probar el algoritmo creado, debemos desarrollar un algoritmo que simule la función del telémetro. Es decir, que dado un entorno virtual y conociendo la pose del robot, sea capaz de calcular el vector de distancias correspondiente al barrido angular de dicho sensor. Para ello, vamos a seguir los siguientes pasos:

[1] Al decir paramétrico, se hace referencia a que los parámetros específicos del telémetro (rango de visión, resolución, etc.) sean fácilmente modificables para dotar al algoritmo de un carácter universal en cuanto a compatibilidad.

[2] abreviatura de radio, por ser originalmente una coordenada polar.

2.2.1. Definir las características del entorno

Necesitaremos conocer la diagonal máxima del recinto y tener dos vectores de coordenadas (uno de X y uno de Y) correspondientes a los obstáculos presentes en el recinto[3]. Ejemplo:

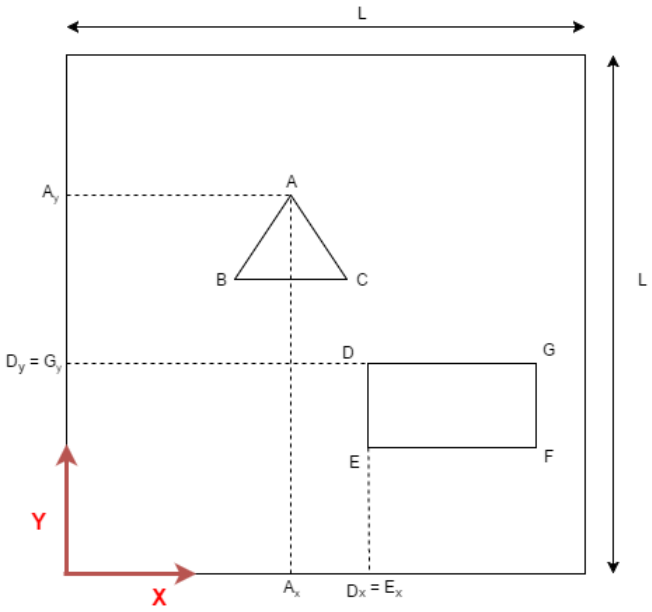


Figura 2.3: Ejemplo de entorno

En este caso, la diagonal máxima será:

$$D_{max} = L * \sqrt{2}$$

y los vectores de obstáculos:

$$OBS_x = \begin{bmatrix} A_x & B_x & C_x & A_x & NaN & D_x & E_x & F_x & G_x & D_x \end{bmatrix}$$

$$OBS_y = \begin{bmatrix} A_y & B_y & C_y & A_y & NaN & D_y & E_y & F_y & G_y & D_y \end{bmatrix}$$

Se usa este formato porque es compatible con la función *polyxpoly* que usaremos a continuación.

2.2.2. Referenciar los obstáculos al sistema de referencia del robot

Para ello usaremos la matriz de transformación ${}^1T_0$, donde si:

$${}^0T_1 * P_1 = P_0 \qquad \text{y} \qquad {}^1T_0 * P_0 = P_1 \qquad \text{entonces} \qquad {}^1T_0 = inv({}^0T_1)$$

[3]el formato de dichos vectores será similar al que se suele usar para dibujar polígonos, es decir, indicando por orden los vértices del obstáculo y repitiendo el primer punto al final, además, se pueden indicar varios polígonos separando sus coordenadas por *NaN's*, el valor vacío de Matlab.

2.2.3. Encontrar los impactos

Se realizan las siguientes operaciones, a través de un bucle, para cada ángulo del barrido:

- Se hace uso de la función *polyxply*, que encuentra intersecciones entre dos polilíneas que serán:
  - Una línea simple correspondiente al rayo de luz emitido en un ángulo determinado. Dicha línea tendrá como longitud la ya definida diagonal máxima del recinto, para así asegurar que salga del recinto y se puedan detectar obstáculos dentro.
  - La polilínea correspondiente al conjunto de líneas de los obstáculos[4].
- En caso de no encontrarse intersecciones, la función devolverá un vector vacío, por lo que se procede a rellenar el valor correspondiente con un NaN(impacto nulo).
- En caso contrario, la función devolverá dos vectores con las coordenadas (X e Y) de las intersecciones encontradas. Para determinar el primer objeto que recibió el impacto, se calculan las distancias del origen a cada una de las intersecciones y se coge la menor, dicha distancia se corresponde al valor del impacto.

Se devuelve el vector de impactos *RAD*, cuyos valores se han ido llenando en cada iteración del bucle.

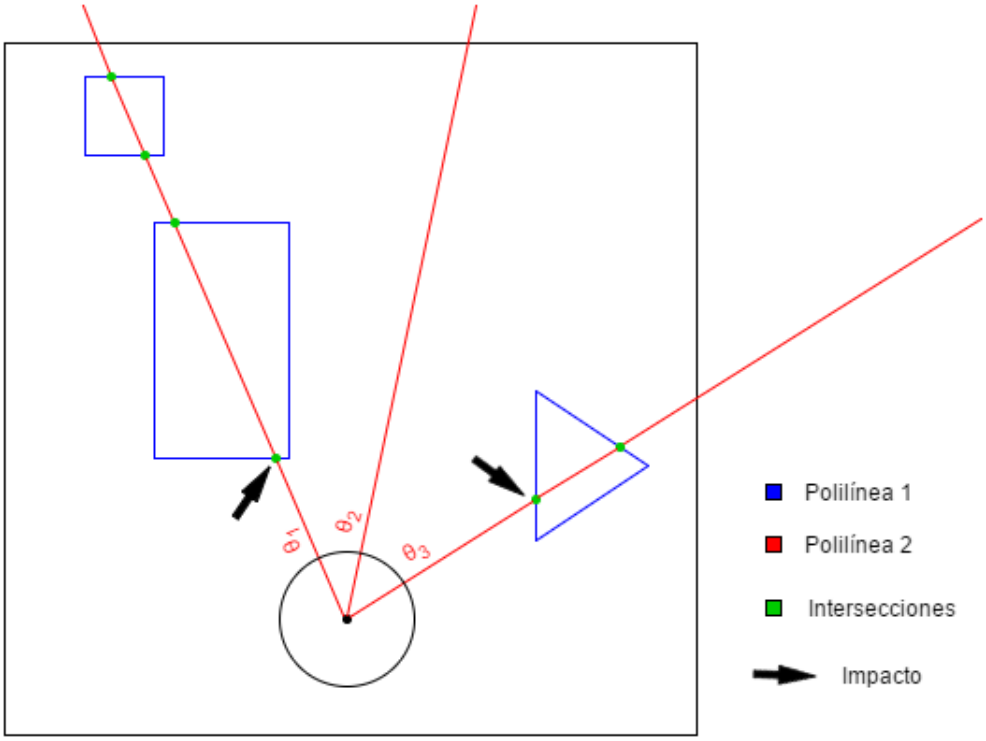


Figura 2.4: Simulación de los impactos

[4]En principio, una polilínea debería ser continua (un solo polígono) pero si usamos NaN's, podemos crear una polilínea múltiple en forma de una polilínea "constante"pero con segmentos vacíos.

La función que implementa el proceso total ha sido llamada VissionSimulation_v1. Dicha función recibe como argumentos de entrada la pose del robot, el valor de diagonal máxima y los dos vectores de obstáculos; y devuelve el vector de impactos *RAD*. El código puede consultarse en el apéndice B.

2.3. Entorno de Simulación

Este será el algoritmo de mayor nivel que haga uso de los dos anteriormente vistos. Sus funciones son:

- Definir el recinto, tanto los bordes como los obstáculos y dibujarlo[5].
- Definir una trayectoria circular para el robot, discretizar un número adecuado de puntos de dicha trayectoria y trabajar con cada uno de ellos de forma sucesiva.
- Para cada punto de la trayectoria, ejecutar el simulador de telémetro y posteriormente con los datos obtenidos, el algoritmo de mapeo *Mapping_v1*.
- Por último, dibujar los datos obtenidos de este ultimo[6], es decir, el mapa.

A continuación se muestra un diagrama de la función, que ha sido llamada *MappingTB_v1*[7]. El código puede consultarse en el apéndice B y los resultados de la simulación en el apéndice A.

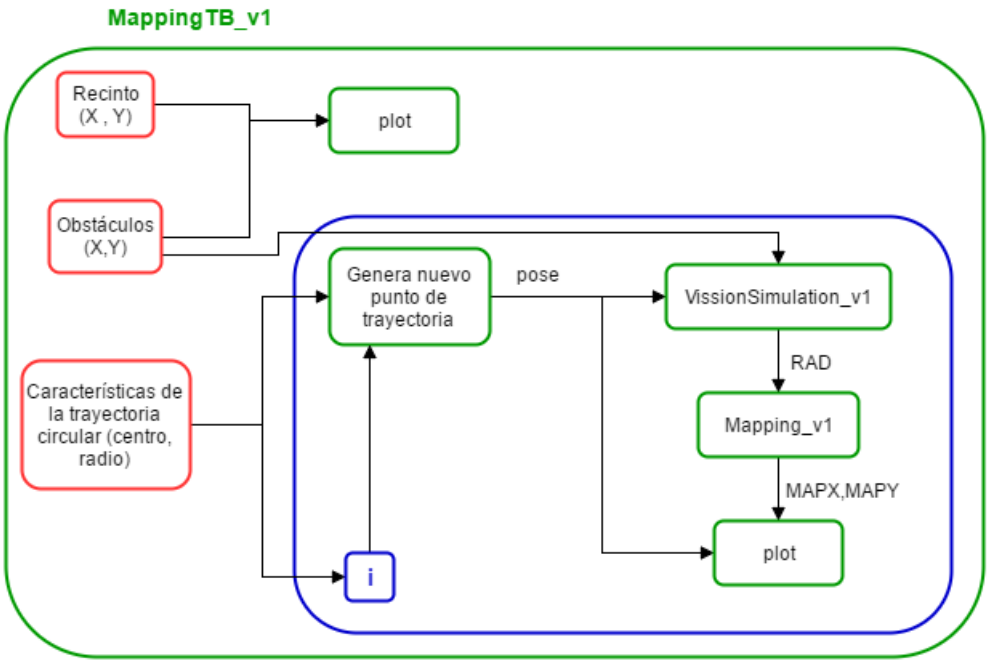


Figura 2.5: Diagrama de *MappingTB_v1*. Definiciones en rojo, funciones/tareas en verde, bucle en azul.

[5] A partir de ahora se usa ese termino para hacer referencia a mostrar datos por pantalla de forma gráfica.

[6] Siempre que se ejecutan funciones de dibujo dentro de bucles, se debe poner un comando de pausa, en nuestro caso se pone uno de 0,1 segundos de pausa.

[7] TB es el acrónimo inglés de *Test Bench*.

2.3.1. Segunda versión

El principal inconveniente de la primera versión es que nunca se llega a tener el mapa global sino que se van mostrando los mapas parciales según se calculan, esto se hizo así ya que el objetivo era probar el correcto funcionamiento de todos sus componentes. Una vez superada esa etapa, modificamos el algoritmo de simulación introduciendo las mejoras pertinentes.

Cambios respecto a la versión anterior:

- Se almacenan los mapas parciales en un vector de mapa global.
- Se cambia el método de dibujo, ahora se dibuja todo con cada iteración, sin mantener nada de una a la siguiente. Además, se hace uso de la función *dibujar_v1*[8], que dibuja un robot con más detalle.
- Se han echo dos funciones con distribuciones diferentes de obstáculos, estas son *MappingTB_v2_1* y *MappingTB_v2_2*[9].

A continuación se muestra el diagrama respectivo, el código se puede consultar en el apéndice B y los resultados de la simulación en el apéndice A.

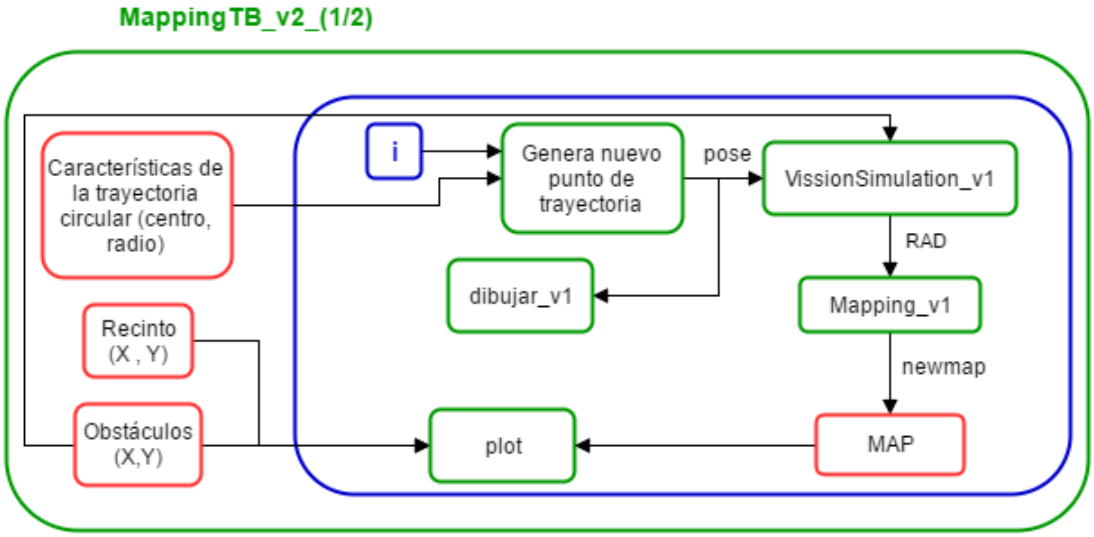


Figura 2.6: Diagrama de *MappingTB_v2*. Definiciones en rojo, funciones/tareas en verde, bucle en azul.

[8]Esta función se detalla en el apéndice C.

[9]De ahora en adelante, el número que se indica después de la versión, hace referencia a una distribución diferente de obstáculos y, en versiones posteriores, a ciertas modificaciones secundarias que se verán en su momento.

2.4. Cinemática del robot

En un robot real, la pose del robot no viene determinada de antemano sino que se debe calcular en función de ciertos parámetros que dependerán del sistema de localización usado. La mayoría de sistemas de localización avanzados requieren de un mapa previo del entorno para funcionar adecuadamente pero ya que en nuestro caso no disponemos de mapas previos, debemos usar un sistema más sencillo que dependa de parámetros fácilmente calculables. El sistema que usaremos para este propósito es el de *Método de localización por odometría*, que calcula la pose en intervalos finitos de tiempo en función de la pose anterior y la velocidad del robot.

EL inconveniente de este sistema es que, al ser un método recursivo, necesitaremos partir de una pose conocida, por eso a partir de ahora, se asume que en el instante inicial el robot se encuentra en el origen de coordenadas[10]. Para determinar la velocidad del robot, se hace uso de sensores en las ruedas del mismo que darán cuenta de la velocidad angular de cada rueda, con estas velocidades y el modelo físico del robot se puede determinar las velocidades lineales de las ruedas y por tanto, la nueva pose en función de estas.

El modelo de robot que estudiamos se conoce como *Modelo de locomoción diferencial*, donde se dispone de dos motores bidireccionales ubicados en el eje central del robot y a ambos lados del centro, siendo simétrica la ubicación de las ruedas respecto al centro del robot.

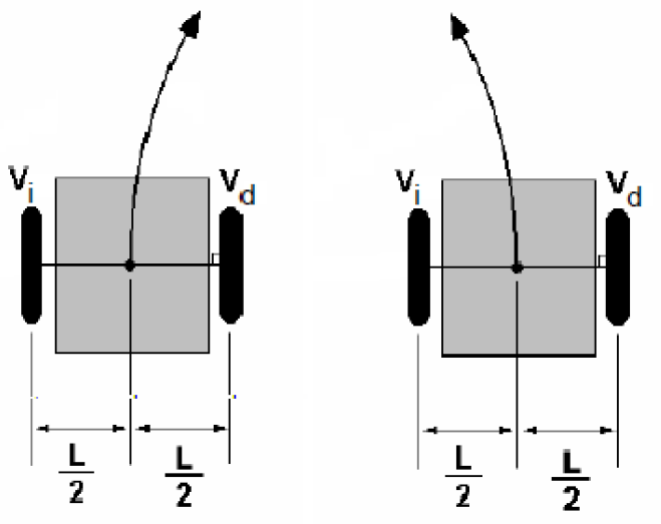


Figura 2.7: Modelo de locomoción diferencial

Para este modelo específico, las fórmulas del modelo cinemático discreto son:

$$v_d = \omega_d \cdot r \tag{2.5}$$

$$v_i = \omega_i \cdot r \tag{2.6}$$

$$v = \frac{v_d + v_i}{2} \tag{2.7}$$

[10]Pose (0,0,0). También se puede interpretar como que el mapa se construirá con referencia a la pose inicial del robot.

$$\omega = \frac{v_d - v_i}{L} \tag{2.8}$$

$$\theta_{k+1} = \theta_k + \omega \cdot \Delta t \tag{2.9}$$

$$x_{k+1} = x_k + v \cdot cos\left(\theta_{k+1} + \frac{\pi}{2}\right) \cdot \Delta t \tag{2.10}$$

$$y_{k+1} = y_k + v \cdot sen\left(\theta_{k+1} + \frac{\pi}{2}\right) \cdot \Delta t \tag{2.11}$$

Donde:

v_d: Velocidad lineal de la rueda derecha [m/s]
v_i: Velocidad lineal de la rueda izquierda [m/s]
ω_d: Velocidad angular de la rueda derecha [rad/s]
ω_i: Velocidad angular de la rueda izquierda [rad/s]
r: Radio de las ruedas [m]
v: Velocidad lineal del robot [m/s]
ω: Velocidad angular del robot [rad/s]
L: Distancia entre las ruedas [m]
Δt: Incremento de tiempo [s]
θ_k: Componente angular de la pose, instante anterior [rad]
θ_{k+1}: Componente angular de la pose, instante posterior [rad]
x_k: Componente X de la pose, instante anterior [m]
x_{k+1}: Componente X de la pose, instante posterior [m]
y_k: Componente Y de la pose, instante anterior [m]
y_{k+1}: Componente Y de la pose, instante posterior [m]

Las dimensiones que utilizaremos son las contempladas por las funciones de dibujo del robot, las cuales son paramétricas ya que dependen del radio que le demos:

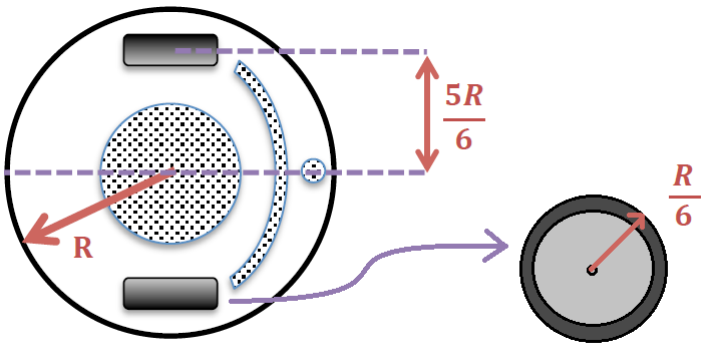


Figura 2.8: Dimensiones del robot

2.4.1. Implementación

Para poder implementar dicho método dentro de nuestra función de mapeo, debemos tener en cuenta que al ser un proceso recursivo, tendremos como variable de entrada la velocidad instantánea con la que habrá que calcular la nueva pose en cada iteración, pero para calcularla necesitaremos conocer la pose anterior, es decir, tener las variables de la pose en un espacio de memoria no volátil, es decir, que se mantenga de una iteración a otra. En Matlab lo que se usan son variables *persistent*.

Por lo expuesto anteriormente, se necesita también de una señal de *Reset* para poder borrar dicha variable estática, normalmente porque se quiera iniciar un nuevo proceso de mapeo, al reiniciar la pose, los valores de la misma se establecerán en la posición de inicio, que como se mencionó anteriormente, será el origen de coordenadas del *Sistema de Referencia Global*.

La función ha sido llamada *Mapping_v2* y los códigos pueden consultarse en el apéndice B.

2.4.2. Entorno de Simulación

Respecto al entorno de simulación, los cambios que se han echo son:

- Se define el incremento de tiempo asociado al muestreo de velocidades angulares.
- Se definen dos vectores con las velocidades angulares instantáneas que se tendrán en las ruedas a lo largo de la simulación.
- Antes de iniciar el bucle de instantes de tiempo, se hace una llamada a la función de mapeo con la señal de *Reset* activa.
- En cada iteración, se calcula la nueva pose, ya que esta información es necesaria para simular el telémetro y dibujar el robot.

La función ha sido llamada *MappingTB_v3*, los códigos pueden consultarse en el apéndice B y los resultados de la simulación en el apéndice A. A continuación se muestra un diagrama del entorno de simulación:

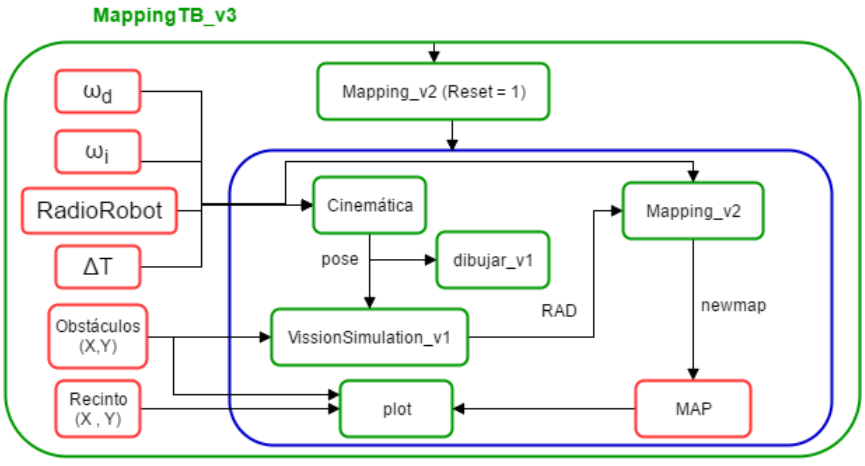


Figura 2.9: Diagrama de *MappingTB_v3*. Definiciones en rojo, tareas en verde, bucle en azul.

Capítulo 3

Introduciendo ruidos

Como era de esperarse, en un entorno real, las cosas no son tan perfectas como han sido hasta ahora sino que existen diversos factores que alteran el funcionamiento ideal del sistema. El factor en concreto que estudiaremos será el ruido de los sensores, tanto del telémetro (error de distancia) como de los sensores de velocidad angular (error de odometría).

Dicho ruido se debe al error del propio sensor y a ruidos que puedan inducirse durante la transmisión de datos desde este al controlador. Se puede modelar como ruido gaussiano, donde:

$$Valor_{real} = Valor_{ideal} + \varepsilon$$

$$\varepsilon = \mu + \sigma \tag{3.1}$$

Donde:

ε: Error total
μ: Media del error, que al ser ruido gaussiano, asumimos cero.
σ: Desviación típica del error, que dependerá de la precisión del sistema sensor-transmisión.

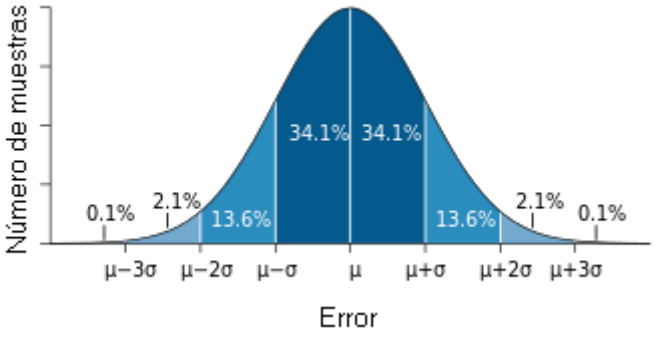


Figura 3.1: Modelo de error gaussiano

3.1. Implementando el ruido en el simulador

Para implementar el modelo de ruido gaussiano en nuestro simulador, necesitaremos hacer una modificación en el algoritmo de mapeo, que es que este devuelva también la pose que ha calculado con las velocidades introducidas. Esto se hace para poder comparar la pose real con la calculada (que contempla los valores no ideales de odometría). La función se ha llamado *Mapping_v3* y su código se puede consultar en el apéndice B.

3.1.1. Entorno de Simulación

Respecto al entorno de simulación, los cambios hechos son:

- Para definir los errores se usa la función *randn*, que genera números siguiendo una distribución normal típica (de media 0 y desviación 1) y multiplicando dicho número por la desviación que vayamos a considerar, a continuación, se suma el valor del error al valor de medida ideal que se tenga. Los valores de desviación se aceptan como variable de entrada al simulador de la forma *MappingTB_v4_1(SigmaOd, SigmaMed)*.

- Respecto al error de medida, se introduce el error como operación intermedia entre la simulación del telémetro y la llamada al algoritmo de mapeo, es decir, se cogen los datos provenientes del simulador y se les suma el error gaussiano para luego introducir el resultado en el algoritmo de mapeo. Los resultados de dicho error se podrán apreciar en el mapa saliente de este.

- Para implementar el error de odometría, se crea dos nuevos vectores de velocidades que son el resultado de sumarle a los vectores originales el error generado con el método descrito en el anterior punto y se introducen estos nuevos vectores como entrada del algoritmo de mapeo. Los resultados se apreciarán en el mapa generado por este.

- Para poder comparar la pose real con la calculada, se calcula la primera aplicando las formulas de cinemática a los vectores de velocidad real (sin error) y la segunda se obtiene del algoritmo de mapeo. Ambas poses se dibujan con un color diferente haciendo uso de la función dibujar_v2[1].

- Se ha considerado la primera llamada al algoritmo de mapeo (donde se resetean las variables estáticas) como primera iteración de la simulación, es decir, que además de resetear la pose, se le pasa el vector *RAD* correspondiente y el mapa resultante se almacena en el mapa global como primeros datos.

A continuación se incluye un diagrama de la función del entorno de simulación, la cual ha sido llamada *MappingTB_v4*. El código puede consultarse en el apéndice B y los resultados de las simulaciones en el apéndice A.

[1] Esta función se detalla en el apéndice C.

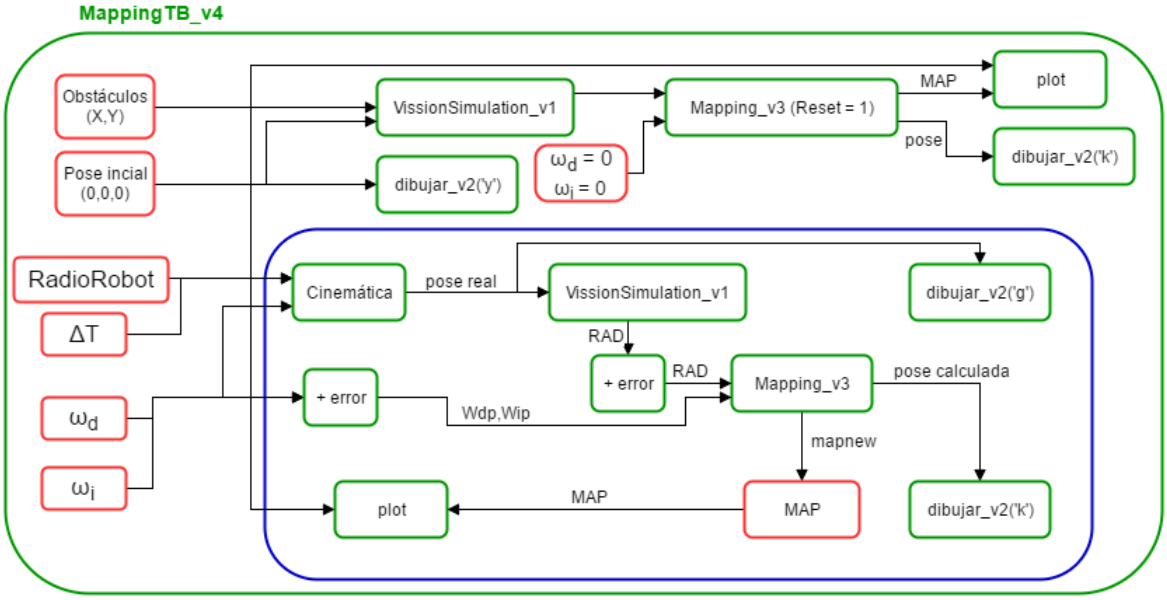


Figura 3.2: Diagrama de *MappingTB_v4*. Definiciones en rojo, funciones/tareas en verde, bucle en azul.

3.2. Corrigiendo el error de odometría

Este error se debe al sensor de velocidad angular de los motores, que ocasionará que la velocidad que recibamos no sea la velocidad real sino que tenga un error añadido del cual se habló anteriormente. Recordemos que la velocidad es usada para calcular la pose en cada iteración, con lo que al tener un error en esta, la pose se verá afectada también, lo cual es crítico ya que nuestra transformación de sistema de referencia del mapa depende únicamente de esta. Además, el calculo de la pose depende también de la pose anterior, por lo que el error de odometría será acumulativo y nuestra pose tendrá mayor error con cada iteración que pase, efecto que se puede apreciar en la tabla A.1 y la gráfica A.7 del apéndice A, las cuales se corresponden al error de pose[2] en cada iteración de una llamada MappingTB_v4_1(0.8,0).En estas podemos observar lo mencionado, que el error es acumulativo y que además, no tiene por qué ir siempre en aumento.

Por lo tanto, debemos diseñar un método de corrección del error que sea iterativo, es decir que se ejecute en cada iteración para corregirlo y así evitar acumulaciones. El método a implementar lo llamamos *Map Matching por fuerza bruta* y consiste en definir una región de fiabilidad alrededor de la pose actual del robot y tomar un número discreto de poses representativas dentro de dicha área(incluyendo la pose original), llamadas poses de prueba. Luego, para cada una de dichas poses(fuerza bruta) se realiza una comparación de mapas entre el mapa que se tiene de anteriores iteraciones y el mapa que se tendría en la nueva pose de prueba, definiendo un parámetro de similitud. Una vez habiendo realizado todas las comparaciones, se toma la mayor similitud y se define dicha pose de prueba como la pose real. Con esto lo que se consigue es, haciendo uso del mapa de anteriores iteraciones, determinar en qué medida los nuevos resulta-

[2] Al ser la pose una variable tridimensional, se ha considerado el error como la distancia tridimensional entre las dos poses, es decir, el módulo del vector que las une.

dos se corresponden con este[3] y comparar dicha medida con los resultados hipotéticos que se obtendrían con poses cercanas a la calculada. Es decir, lograríamos corregir el error de pose en cierta medida siempre y cuando la región de fiabilidad esté correctamente definida. Seguiremos hablando de esto más adelante, cuando implementemos dicha lógica en el apartado 3.2.2.

3.2.1. *Occupancy Grid*

El primer paso para acometer nuestro objetivo es definir e implementar lo que en robótica se conoce como *Occupancy Grid*, una forma de definir un mapa que como su nombre indica, define una rejilla de posiciones dentro del recinto, donde puede o no haber algo, en nuestro caso obstáculos. Es decir que lo que haremos será obtener un mapa digital de posiciones discretizadas de valor en formato binario con su respectiva resolución. Esto se hace porque al tener un mapa discreto con un número finito de resultados es mucho más sencillo comparar dos de estos.

La función implementada que genera la *Occupancy Grid* se ha llamado *CalculateMAP_v1* y su funcionamiento se detalla al completo en el apéndice C. A continuación se citan sus principales características:

- Recibe como argumentos de entrada la longitud del recinto cuadrado(*LongRes*), la resolución de la rejilla(*Incremento*), la pose(*x,y,theta*) y el vector de impactos de telémetro (*RAD*).
- Devuelve el mapa discreto o *Occupancy Grid* en forma de matriz bidimensional con elementos en formato binario. El formato de la rejilla es compatible con la función de Matlab llamada *Mapshow*, cuya función es dibujar el mapa.
- Al igual que su antecesora analógica, tiene definidas como variables internas los parámetros correspondientes al telémetro, pudiendo modificarse según la necesidad.

3.2.1.1. Implementación

El algoritmo de mapeo que implemente la *Occupancy Grid* deberá tener las siguientes características:

- Ya que muy pronto implementaremos el *map matching*, vamos a empezar a guardar el mapa global en esta función y no en el entorno de simulación. Para ello se define el mapa como una variable estática, para que se mantenga de una iteración a otra, al igual que ya se había hecho con las variables de la pose.
- Para poder ejecutar la función *CalculateMAP_v1* necesitamos la longitud del recinto cuadrado y la longitud de cada celda de la *Occupancy Grid*. La primera se aceptará como variable de entrada para definirla en el entorno de simulación, pero el incremento (longitud de celda) se define dentro de esta función, por defecto lo establecemos a $0,1$.
- Tiene una variable de reseteo, que pone todas las variables estáticas a 0, incluido el mapa.

[3]Claramente, habrá parte de los resultados que se correspondan a una nueva zona del mapa no vista anteriormente, pero son las muestras correspondientes a zonas conocidas del mapa las que determinarán el grado de correspondencia.

- Sus tareas son calcular la nueva pose, llamar a la función *CalculateMAP_v1*, almacenar la información nueva en el mapa global y por último, devolver este mapa como argumento de salida.

- Para introducir los nuevos datos en el mapa global sin afectar los mapas anteriores se hace uso del operador OR, ya que al ser matrices de 1's y 0's, este operador conservará tanto los impactos antiguos como los nuevos.

- Las variables de entrada y salida de la función son similares a la versión anterior con dos diferencias:

  - Se pide también como argumento de entrada la longitud del recinto cuadrado.
  - El mapa de salida ya no son dos vectores de coordenadas sino una matriz bidimensional con el mapa discreto global.

la función ha sido llamada *Mapping_v4* y el código pude consultarse en el apéndice B.

3.2.1.2. Entorno de Simulación

Los cambios que presenta frente a la versión anterior son:

- Al momento de llamar al mapeador, se le pasa también la longitud del recinto como argumento de entrada.

- El mapa recibidos del mapeador es una matriz que representa al mapa global, por lo que, a diferencia que la versión anterior donde debíamos almacenar los mapas parciales en un mapa global, ahora únicamente hay que dibujar el mapa recibido en cada iteración.

- Para dibujar el mapa matricial se hace uso de la función *mapshow* que necesita conocer ciertos parámetros del mapa que va a dibujar. Dichos parámetros se le pasan en una estructura previamente definida (*R*) haciendo uso de la función *maprasterref*, estos parámetros son los límites superior e inferior de cada eje y las dimensiones de la matriz de celdas. Además, la función *mapshow* interpreta los datos de la matriz como valores de luminancia, es decir, como la cantidad de blanco sobre fondo negro, siendo el 1 el valor de luminancia máximo (color blanco) y el 0 el valor de luminancia mínimo (color negro). Por esto, si queremos dibujar el mapa como obstáculos negros sobre fondo blanco, debemos invertir la matriz del mismo, al ser una matriz de tipo lógico, podemos hacer esto fácilmente con el operador NOT.

- Por problemas de dibujo, se ha cambiado la técnica de dibujar los obstáculos, pasando del anterior *mapshow* de polígonos a un *plot* estándar.

- En *MappingTB_v5_2* se ha cambiado además del mapa, la base de tiempos, aumentando el periodo de muestreo a 0,5s y usando velocidades angulares considerablemente menores[4]. Debe tenerse en cuenta a la hora de introducir el error de odometría que para esta versión debe ser menor para conservar la relación de ruido[5].

[4]Este cambio se conserva para todas las segundas subversiones siguientes.
[5]Relación entre el valor medio de una señal y la desviación típica de su error.

A continuación se muestra el diagrama de la función. Los códigos pueden consultarse en el apéndice B y los resultados de la simulación en el apéndice A, all no haber modificado nada referente a la odometría, no se detalla la tabla de errores.

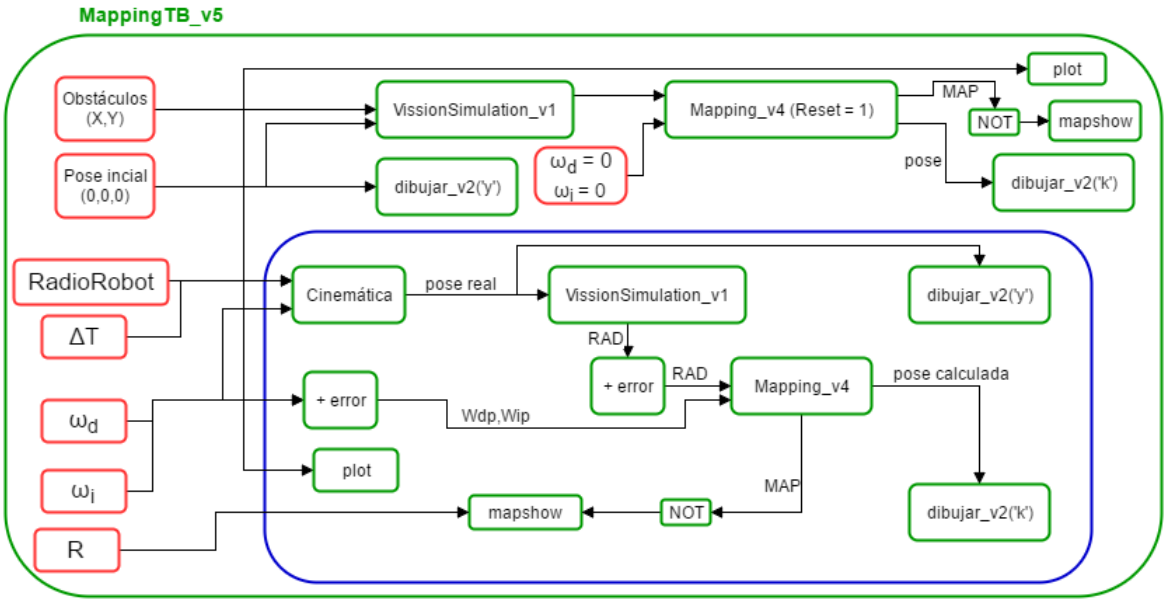


Figura 3.3: Diagrama de *MappingTB_v5*. Definiciones en rojo, funciones/tareas en verde, bucle en azul.

3.2.2. *Map Matching*

Una vez implementada de forma correcta la *Occupancy Grid*, tenemos el camino libre para efectuar el proceso conocido como *Map Matching*. Como ya se mencionó, este método intenta contrarrestar los errores de odometría mediante la comparación de los datos recibidos del telémetro con los datos almacenados de anteriores iteraciones, que se asumen como válidos. Por lo tanto, si asumimos que en la primera iteración la pose es conocida, los datos provenientes del sensor podrán tomarse como válidos sin necesidad de hacer comprobaciones y serán el punto de partida para las siguientes comprobaciones recursivas. Así, si nos aseguramos de que los resultados de aplicar este método en cada iteración sean válidos, tendremos con cada iteración nuevos datos válidos que se usarán para comprobar los datos de la siguiente iteración y así sucesivamente hasta terminar el ciclo.

Bien, una vez definida la idea general del desempeño de este método, vamos a profundizar un poco más en el funcionamiento del mismo, concretamente, en la comprobación y corrección de datos.

Nos enfrentamos a una situación donde tendremos dos mapas, uno de ellos será el mapa global almacenado, que dijimos que asumiríamos como válido y el otro es el mapa nuevo procedente de la nueva lectura del telémetro. Este último podrá ser o no ser más o menos correcto dependiendo del error de odometría que se haya dado, de ser totalmente correcto, la mayoría de lecturas se corresponderían con las almacenadas en el mapa global y habría también un número de lecturas nuevas, correspondientes a las áreas del recinto no antes vistas; por el contrario, de tener un mapa con error de odometría, lo que tendremos será un mapa donde las lecturas que deberían corresponderse al mapa global, estarán desfasadas respecto a este, ya que el error de odometría

habrá afectado a la pose conocida y esta a su vez, a la transformación de sistema de referencia.

La forma correcta de corregir este error sería encontrar con exactitud el desfase entre ambos mapas[6] para así poder determinar la pose real pero es un problema de gran complejidad que además, debería tener en cuenta aquellas medidas que no se corresponden con el mapa global por ser nuevas áreas visibles. La solución que se plantea sigue la misma línea deductiva pero es de una complejidad considerablemente menor. La idea es probar con diferentes poses al rededor de la conocida y generar para cada una, el mapa de los últimos datos recibidos. Una vez tengamos este conjunto de mapas de prueba, los compararemos con el mapa global almacenado y elegiremos el que mejor se corresponda a este. Luego, daremos como válida la pose que dio origen al mapa elegido y almacenaremos los nuevos datos referenciados a la pose corregida. En el caso de no haber error, el mapa elegido como mejor correspondido será el referente a la pose calculada en primera instancia, que claramente debe estar dentro de las poses de prueba. Cabe destacar que los datos nuevos que no se corresponden a ningún obstáculo presente en el mapa almacenado no afectan a esta comparación, ya que a la hora de calcular la correspondencia entre mapas solo se toman como validos los puntos comunes y el resto se descartan[7]. A esta área alrededor de la pose conocida se ha llamado *ventana de comparación* y de su correcta definición dependerá el éxito del método. Por no incurrir en demasiado coste computacional, hemos definido la ventana con tres valores diferentes para cada dimensión de la pose, la central y dos equidistantes a cada lado de esta; por lo que tendremos un total de 27 poses de prueba, incluida la original.

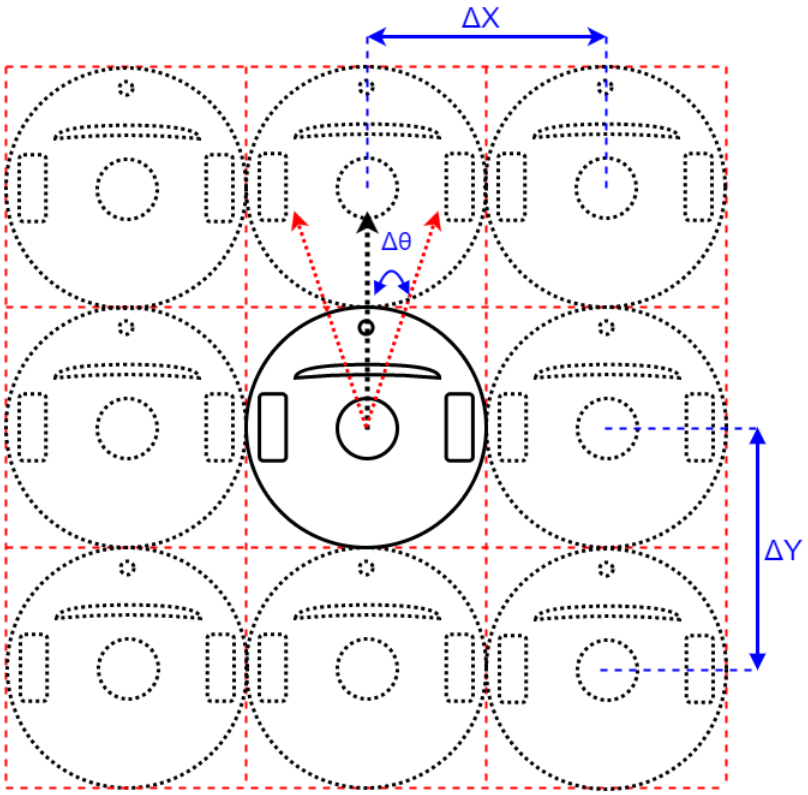


Figura 3.4: Ventana de comparación de mapas

[6]Nótese que el desfase tiene tres grados de libertad correspondientes a las tres variables de la pose.

[7]Por el contrario, estos datos sí que podrían incurrir en falsas comparaciones al confundirse con otro obstáculo anteriormente visto.

Los parámetros correspondientes a las dimensiones de la *ventana de comparación* Δx, Δ y $\Delta\theta$ los definiremos como la variación típica de dichas variables debidas al error de odometría, para ello haremos uso del método matemático de propagación del error. Con este método seremos capaces de calcular en qué medida afectan los errores de las variables de las cuales depende una tercera, en el error de esta. Para la variable p que depende de las variables $(q_1, q_2, q_3....q_n)$:

$$\Delta p = \sum_{i=1}^{n} \left| \frac{\partial p}{\partial q_i} \right| \Delta q_i \tag{3.2}$$

Donde Δ representa la variación o error de una variable de forma que:

$$p = \overline{p} \pm \Delta p \tag{3.3}$$

Vamos a aplicar este método matemático a las ecuaciones de la cinemática de robots con locomoción diferencial para así determinar la propagación de los errores de velocidad en la pose. Desarrollando las fórmulas 2.5, 2.6, 2.7, 2.8, 2.9, 2.10 y 2.11 obtenemos:

$$\theta_{k+1} = \theta_k + (\omega_d - \omega_i) \cdot \frac{r \cdot \Delta t}{L} \tag{3.4}$$

$$x_{k+1} = x_k + (\omega_d + \omega_i) \cdot \frac{r \cdot \Delta t \cdot cos\left(\theta_{k+1} + \frac{\pi}{2}\right)}{2} \tag{3.5}$$

$$y_{k+1} = y_k + (\omega_d + \omega_i) \cdot \frac{r \cdot \Delta t \cdot sen\left(\theta_{k+1} + \frac{\pi}{2}\right)}{2} \tag{3.6}$$

Entonces, aplicando la fórmula de propagación del error 3.2 a 3.4:

$$\Delta\theta_{k+1} = \left| \frac{\partial \theta_{k+1}}{\partial \theta_k} \right| \Delta\theta_k + \left| \frac{\partial \theta_{k+1}}{\partial \omega_d} \right| \Delta\omega_d + \left| \frac{\partial \theta_{k+1}}{\partial \omega_i} \right| \Delta\omega_i$$

Si asumimos que el valor anterior θ_k es exacto, $\Delta\theta_k = 0$. Además, $\Delta\omega_d = \Delta\omega_i = \Delta\omega$:

$$\Delta\theta_{k+1} = \cancel{\left| \frac{\partial \theta_{k+1}}{\partial \theta_k} \right| \Delta\theta_k} + \frac{r \cdot \Delta t}{L} \cdot \Delta\omega + \frac{r \cdot \Delta t}{L} \cdot \Delta\omega$$

$$\Delta\theta_{k+1} = \frac{2 \cdot r \cdot \Delta t}{L} \cdot \Delta\omega \tag{3.7}$$

De forma análoga, aplicamos 3.2 a 3.5 y 3.6:

$$\Delta x_{k+1} = \left(r \cdot \Delta t \cdot cos\left(\theta_{k+1} + \frac{\pi}{2}\right)\right) \Delta\omega + \left(\frac{(\omega_d + \omega_i) \cdot r \cdot \Delta t}{2} \cdot sen\left(\theta_{k+1} + \frac{\pi}{2}\right)\right) \Delta\theta_{k+1} \tag{3.8}$$

$$\Delta y_{k+1} = \left(r \cdot \Delta t \cdot sen\left(\theta_{k+1} + \frac{\pi}{2}\right)\right) \Delta\omega + \left(\frac{(\omega_d + \omega_i) \cdot r \cdot \Delta t}{2} \cdot cos\left(\theta_{k+1} + \frac{\pi}{2}\right)\right) \Delta\theta_{k+1} \tag{3.9}$$

Así, tendremos un algoritmo inteligente que calculará las dimensiones de la ventana en cada iteración en función de la pose actual y la desviación típica de las velocidades angulares. En nuestro caso, dicho valor podemos obtenerlo del entorno de simulación pero en un caso real, se tendría que modelar el error de odometría para calcular su desviación típica o en su defecto, encontrar empíricamente un valor con el que el algoritmo funcione correctamente.

La forma de calcular la similitud entre dos mapas será mediante el número de elementos en común, es decir, que ambos mapas tengan la misma celda definida como ocupada por un obstáculo.

3.2.2.1. Implementación

Para implementar este método en un algoritmo computacional se han tenido en cuenta las siguientes consideraciones:

- En la primera iteración, cuando la variable de *Reset* está a '1', se inicializan las variables estáticas incluido el mapa. Además, al estar en una pose conocida, no se hace ningún matching sino que se almacenan directamente los datos provenientes del sensor en el mapa global.
- Para generar la *Occupancy Grid* se hace uso de la nueva versión *CalculateMAP_v2* que introduce ligeras mejoras de rendimiento. Esta función se detalla en el apéndice C.
- Los parámetros de la ventana se calculan con las fórmulas 3.7, 3.8 y 3.9 inmediatamente después del cálculo cinemático de la pose, ya que algunos de ellos dependen del valor actualizado de esta.
- La separación entre ruedas del robot se sigue considerando como $\frac{10*R}{6}$, siendo R el radio paramétrico del robot.
- Para abarcar todas las poses de prueba, se hace uso de tres bucles anidados, uno por cada dimensión de la pose.
- Dentro de dichos tres bucles se realiza el cálculo del mapa de prueba (*MAPprobe*) y su comparación con el mapa global (*MAPpersistent*). Dicha comparación se hacía en un principio con dos bucles anidados que recorrían cada posición de las matrices en busca de elementos comunes pero se cambió el método de comparación por el de calcular la media de los elementos de la matriz resultante de la operación AND[8] de ambas matrices (prueba y global). Este nuevo método tiene un mejor rendimiento, hecho que se demuestra en el apéndice D. El método descartado se encuentra en el código de la función en forma de comentarios.
- Para identificar la mejor coincidencia dentro del triple bucle se utiliza el índice *w*, que da cuenta del número de comprobación que se está haciendo. Cuando se detecta que una comprobación es mayor que la máxima hasta ese momento, se usa el índice *w* para definir el vector de ceros *winner* donde el único '1' será el correspondiente al elemento *w*. Una vez terminadas todas las comprobaciones, se realiza nuevamente tres bucles anidados con un índice común *w*. Dentro de estos, se comprueba si el elemento *w* del vector

[8]Este operador es ideal, ya que su resultado será '1' solo cuando ambas matrices tengan un '1' en dicha 'posición. Además, se calcula la media de elementos como forma de cuantificar la cantidad de 1's de dicha matriz.

winner es '1' y de ser así, se toma esa pose como válida, se calcula la *Occupancy Grid* correspondiente y se interrumpe el triple bucle.

- Como en versiones anteriores, luego de calcular la *Occupancy Grid* de la iteración actual, se almacena dichos datos en el mapa global haciendo uso del operador OR.

La función implementada ha sido llamada *Mapping_v5* y el código puede consultarse en el apéndice B.

3.2.2.2. Entorno de Simulación

El entorno de simulación implementado para esta nueva versión del algoritmo de mapeo no tiene ningún cambio respecto a la versión anterior ya que todas las consideraciones necesarias para el matching ya han sido tenidas en cuenta. Se cambia nuevamente el color de dibujar la pose real y a la hora de llamar al mapeador, se le pasa también la desviación típica de odometría. El protagonista de esta nueva versión es el mapeador, que tendrá que ser capaz de anticiparse a posibles errores de odometría para corregir la pose y así tener un mapa correcto.

La función ha sido llamada *MappingTB_v6* con dos subversiones correspondientes a diferentes entornos con diferente base de tiempo al igual que su versión anterior. El código puede consultarse en el apéndice B y los resultados de la simulación en el apéndice A.

3.2.2.3. Evaluación de Resultados

Como podemos observar en los resultados expuestos en el apéndice A, el algoritmo parece responder correctamente a nuestras necesidades. Con errores considerables ($\sigma_{od} = 1,1 rad/s$) logra anticiparse y corregir la pose, obteniendo mapas muy buenos. En las tablas y gráficas observamos que los errores de pose son relativamente pequeños y no acumulativos comparados a la versión anterior. Además, al tener una *ventana de comparación* de dimensiones dinámicas, podemos hacer que se adapte perfectamente a errores de distinta desviación típica como podemos observar, se han simulado errores de 0.5, 0.8 y 1.1 rad/s de desviación típica y de 0.16 y 0.3 rad/s para la segunda subversión que trabaja con velocidades más pequeñas.

El problema de este algoritmo es cuando el error resulta mucho más grande que la desviación, porque aunque se corrige en parte la pose, sigue habiendo un error que en siguientes iteraciones se considera nulo y esto puede incurrir en acumulaciones de error.

Por otro lado, al depender siempre del mapa global acumulado en memoria, es importante que el robot no sufra cambios bruscos en su rango de visión, es decir, que siempre se tenga un porcentaje de este que corresponda a datos ya vistos anteriormente para poder tener una referencia a la hora de corregir la pose. En caso de haber un cambio brusco, como en giros en esquinas, lo que puede suceder es que al no tener una referencia sólida, no se corrija correctamente la pose y se almacenen los obstáculos desplazados de su posición original. Al tener un mapa global erróneo, será imposible corregir la pose hasta que no se vuelva a tener visión de los obstáculos con referencia correcta. Es decir que para un entorno/trayectoria como el de *MappingTB_v6_1*, si se llega a este caso en una esquina, los nuevos datos se almacenarán desplazados y la pose no se corregirá hasta volver a pasar por los primeros obstáculos divisados y aunque entonces se corrija la pose, los datos erróneos almacenados permanecen en la memoria.

Respecto al ruido de medida, observamos que puede engañar al matching, pero al tener valor medio 0, no llega a afectar considerablemente a la corrección de la pose siempre y cuando no sobrepase ciertos límites de desviación típica.

En conclusión, tenemos un algoritmo de actuación frente a ruidos de odometría que funciona bien bajo ciertas condiciones:

- Partir de una pose conocida.
- Que durante la trayectoria se tenga siempre cierto porcentaje de visión de obstáculos ya vistos anteriormente para usarse como referencia o en su defecto, pasar varias veces por las mismas zonas.
- Tener un error de odometría de tipo gaussiano con un valor de desviación típica conocido ya sea mediante modelización del mismo u otros.
- El error de medida debe ser de desviación considerablemente pequeña para no engañar al matching.

La principal debilidad de este algoritmo es el hecho de que almacene los datos de forma definitiva, es decir, que no sea capaz de "olvidar"datos erróneos o no permanentes, como sería el caso de obstáculos móviles. Este problema se intenta resolver a continuación, junto con la corrección del error de medida.

3.3. Corrigiendo el error de medida

Este error se debe al sensor de distancia o telémetro láser, el cual nos dará un valor de distancia a los obstáculos no exacto ya sea por propio diseño interno de este o por interferencias en la transferencia de información. A diferencia del error de odometría, este afecta directamente al mapa, ya que como hemos visto en las simulaciones, al almacenar diferentes impactos cercanos a un punto pero con un error añadido, se produce un ensanchamiento del obstáculo, siendo su área en el mapa mayor que la real.

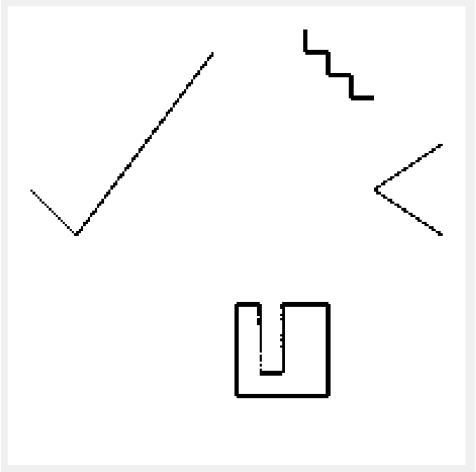


Figura 3.5: Mapa de *MappingTB_v5_2* sin error de distancia

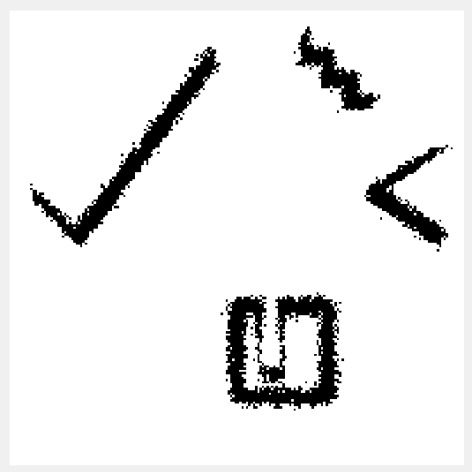


Figura 3.6: Mapa de *MappingTB_v5_2* con error de distancia 0.2

Para corregir este error necesitamos que el algoritmo sea capaz de guardar solo las medidas que más se repitan y olvidar el resto. Este planteamiento solucionaría de igual forma el problema planteado de los obstáculos móviles y las falsas medidas. La solución propuesta para alcanzar este objetivo es implementar una *Occupancy Grid probabilística* cuyos valores puedan estar en un rango no booleano. Así, no se tendría para cada celda un valor de '0' para indicar que no hay obstáculo y un valor de '1' para indicar que lo hay sino que se tendría un número entre 0 y 100, por ejemplo, que determinaría la probabilidad de que en dicha celda se encuentre un obstáculo. Con esto lograríamos que mientras mas veces almacenemos el mismo impacto, mayor sea su probabilidad, y mientras más veces veamos una celda como no ocupada, menor sea su probabilidad; logrando así el objetivo propuesto.

La función implementada que calcula y genera la *Occupancy Grid propabilística* se ha llamado *CalculateMAP_v3* y su funcionamiento se detalla al completo en el apéndice C. A continuación se detallan sus características principales:

- Además de los argumentos de entrada de la versión anterior, recibe también una matriz correspondiente al mapa anterior sobre el que se efectuarán las modificaciones pertinentes para generar el nuevo mapa.
- No acota el rango de valores que puede tomar cada celda del mapa, sólo realiza incrementos y/o decrementos en función de la información recibida.
- Incrementa la celda correspondiente al punto de impacto, decrementa las celdas correspondientes a la trayectoria del haz de luz hasta el punto de impacto, decrementa las celdas correspondientes a la trayectoria del haz de luz hasta el borde del recinto en caso de recibir un impacto nulo.
- Se define como parámetros internos los correspondientes al telémetro así como la magnitud de los incrementos y decrementos correspondientes a las distintas situaciones.

3.3.1. Implementación

Una vez tenemos la función responsable de modificar el mapa en función de la información del sensor, vamos a implementar la función de mapeo con el resto de características necesarias:

- Las variables de entrada y salida permanecen inalteradas respecto a *Mapping_v4*.
- Ya que nuestro objetivo es probar la nueva *Occupancy Grid probabilística*, vamos a dejar de lado el *matching*, por ahora.
- Al igual que en versiones anteriores, tendremos variables estáticas y aplicaremos las fórmulas de la cinemática para calcular la pose de forma recursiva.
- Debemos definir el rango de valores en el que estarán las probabilidades del mapa, este se ha tomado de 0 a 100, por lo tanto:
    - Al inicializar el mapa, le daremos a todas las celdas una probabilidad media de 50, ya que es la mejor forma de representar la incertidumbre inicial.

- Cada que modificamos el mapa mediante *CalculateMAP_v3* debemos asegurarnos de que ninguna celda haya superado los límites establecidos. Por convenio, aquellos valores que salgan del límite se fuerzan a este[9].
- Antes de sacar el mapa como resultado final, lo escalaremos en un factor 1:100, es decir, que el mapa resultante tendrá valores en el rango [0;1]. Esto se hace para facilitar su compatibilidad con la función *mapshow*.

La función ha sido llamada *Mapping_v6* y el código se puede consultar en el apéndice B.

3.3.2. Entorno de Simulación

Los cambios introducidos respeto a la versión anterior son:

- A la hora de llamar al mapeador, lo hace a la nueva versión *Mapping_v6*.
- Para poder dibujar correctamente el mapa[10] y debido a que la función *mapshow* interpreta valores de luminancia, se debe invertir los valores de este por medio de la operación $1 - x$.[11]
- A parte de las dos subversiones que se tenían, se ha implementado una tercera cambiando los obstáculos y el recorrido del robot, la base de tiempo permanece similar a la segunda subversión. La novedad que presenta esta nueva subversión son los obstáculos móviles, que se han conseguido modificando el vector de obstáculos con cada nueva iteración. Así, esta versión pondrá a prueba las nuevas características de nuestro mapeador, concretamente, la capacidad de olvidar datos no constantes.

La función implementada ha sido llamada *MappingTB_v7* y su código puede consultarse en el apéndice B. Se ha creado también *MappingTB_v6_3*, que aplica la versión anterior del algoritmo de mapeo (*Mapping_v5*) al mapa de obstáculos móviles, esto se hace para poder comparar las nuevas características frente a obstáculos móviles del nuevo mapeador. Todos los resultados obtenidos de la simulación se pueden consultar en el apéndice A, ya que el error que estamos analizando no afecta a la pose, se omiten las tablas de error cuadrático en esta.

3.3.3. Evaluación de Resultados

Como hemos podido observar en los resultados expuestos en el apéndice A, la *Occupancy Grid probabilística* contiene mucha más información que la booleana y responde bastante bien a nuestras demandas.

Por un lado, como podemos observar en los resultados de *MappingTB_v7_1* y *MappingTB_v7_2*, contrarresta de forma adecuada el error de medida, ya que al final las celdas que quedan con mayor probabilidad son aquellas que más veces se visualizó, mientras que los datos que aparecen

[9] Si estamos completamente seguros (100) de un obstáculo y lo volvemos a ver, seguimos estando completamente seguros (100) y de forma similar para el 0.

[10] Como tendremos valores de probabilidad, el 0 debe corresponderse a probabilidad nula de encontrarse un obstáculo en dicha celda y deberá visualizarse de color blanco. La celda de valor 1 corresponde a probabilidad absoluta de contener un obstáculo y deberá visualizarse en negro. Todos los valores intermedios deberán seguir esta lógica acercándose al negro en cuanto mayor sea su probabilidad.

[11] Antes, al tener únicamente valores lógicos, se usaba el operador NOT.

poco (usualmente debidos al error) terminan con una probabilidad baja. Así, observamos que el resultado final es un buen modelo de la realidad, ya que las celdas con mayor probabilidad (colores mas cercanos al negro) son aquellas que se corresponden al borde de los objetos y las celdas de los alrededores de estos tienen mayormente probabilidades bajas (colores cercanos al blanco). En la siguiente figura podemos observar una comparación con la versión anterior para simulaciones con error de desviación 0.2. Recordar que 0.2m es un error bastante grande y con telémetros láser es difícil que lleguemos a esta cifra.

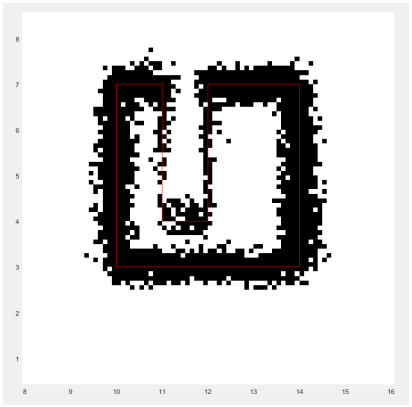


Figura 3.7: Versión anterior

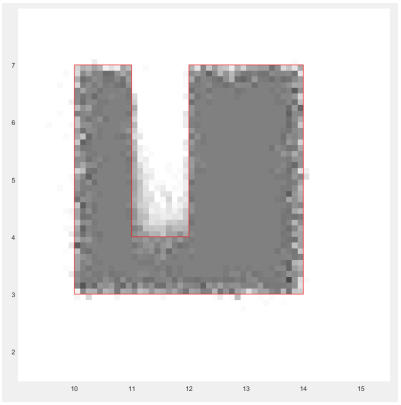


Figura 3.8: Nueva versión

Por otro lado, y gracias a las simulaciones de *MappingTB_v7_3* y su comparación con las de *MappingTB_v6_3*, podemos observar que se ha logrado el comportamiento que buscábamos acerca de los obstáculos no permanentes. Estos se olvidan con el tiempo ya que su probabilidad disminuye con cada iteración que no se visualizan; cosa que no sucede con la anterior versión, donde se guardan todos los datos recibidos.

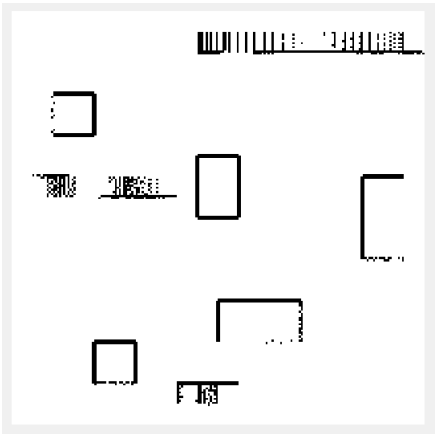


Figura 3.9: Versión anterior

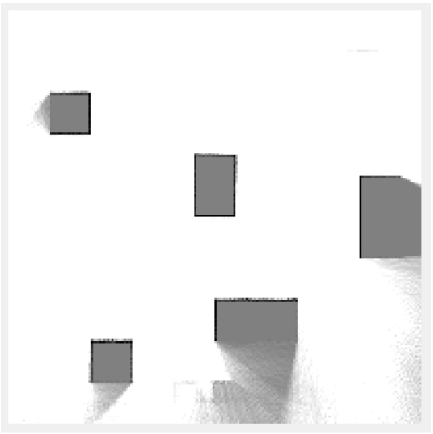


Figura 3.10: Nueva versión

Por último, observamos que el interior de los obstáculos permanece inalterado a la probabilidad media con la que inicializamos el mapa al igual que las zonas no vistas. Esto es una ventaja también, ya que el mapa es mucho más fiable y da cuenta también de las zonas de incertidumbre.

Capítulo 4

Versiones Finales

Llegado a este punto, hemos logrado corregir los dos errores propuestos de forma independiente con distintos métodos. Ahora intentaremos unir ambos métodos en un solo algoritmo para dar paso a la versión final del mapeador.

4.1. Versión 7 del Algoritmo de Mapeo

Esta versión implementa tanto la *Occupancy Grid probabilística* como el *map matching*. Para ello, partimos de la versión 6 (*Occupancy Grid probabilística*) y sobre ella agregamos los procedimientos correspondientes al *map matching* implementados en la versión 5. La *ventana de comparación* se seguirá calculando del mismo modo, por propagación del error de odometría pero habrá que cambiar el método de comparación entre el mapa global y el mapa de prueba ya que ahora tenemos un nuevo formato de mapa.

De esta forma, recurrimos al método de comparación al que hemos llamado *comparación híbrida por máscara*. Híbrida porque utilizaremos la configuración booleana para los mapas de prueba y la configuración probabilística para el mapa global. Así, usaremos el mapa de prueba como máscara del mapa probabilístico, es decir que usaremos las posiciones que en el mapa booleano tengan un valor verdadero (obstáculo detectado) para buscar los valores de sus celdas equivalentes en el mapa probabilístico. El valor medio de estas celdas debería dar cuenta del grado de correspondencia entre ambos mapas.

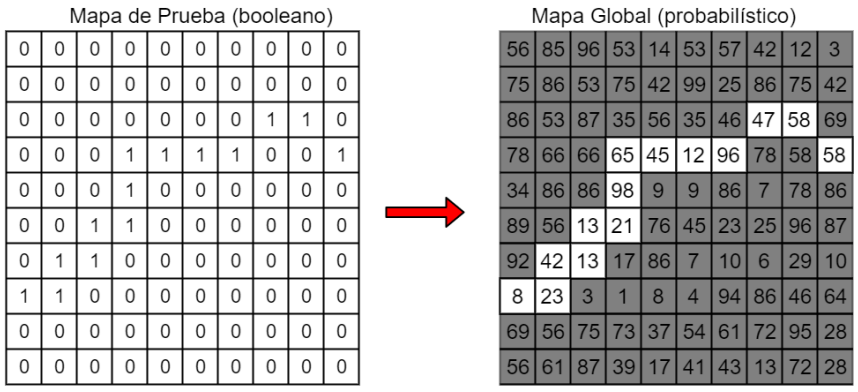


Figura 4.1: Diagrama de la *comparación híbrida por máscara*

4.1.1. Implementación

Tal como se mencionó, partimos de la función *Mapping_v6* y agregamos las operaciones referentes al matching de *Mapping_v5*. los cambios introducidos son:

- Se inicializa el mapa con valores de 50 y al recibir la señal de *reset*, se considera la primera iteración con la pose conocida por lo que se genera directamente el nuevo mapa a partir de la información del telémetro.
- Para calcular el mapa de prueba dentro del bucle triple se utiliza la función *Calculate-MAP_v2*.
- Para comparar el mapa de prueba con el mapa global se aplica lo que en Matlab se conoce como matriz de máscara que trata de usar una matriz de tipo lógico[1] para determinar los indices de los elementos que se extraerán de una segunda matriz para colocarse en un vector lineal. Luego de realizar esto, hallamos el valor medio del vector resultante y lo almacenamos en nuestra variable de coincidencia.
- Una vez definida la pose corregida, se calcula el nuevo mapa probabilístico a partir de esta y de la información recibida por el telémetro haciendo uso de la función *Calculate-MAP_v3*.

La función implementada ha sido llamada *Mapping_v7* y el código se puede consultar en el apéndice B.

4.1.2. Entorno de Simulación

No representa ningún cambio respecto a la versión anterior. La función ha sido llamada *MappingTB_v8* y cuenta con dos subversiones[2] similares a anteriores versiones. Los códigos se pueden consultar en el apéndice B y los resultados de las simulaciones en el apéndice A.

4.1.3. Evaluación de Resultados

Como hemos podido observar en los resultados expuestos en el apéndice A, el *mathcing* no está haciendo su trabajo correctamente. Independientemente de la subversión y del error que le introduzcamos a la simulación, el error no se está corrigiendo iterativamente como debería, lo que produce errores acumulados con gráficas de tendencia creciente. Además, como observamos en la figura A.59, son pocas las veces que el error cuadrático medio resulta considerablemente bajo como para aceptar el funcionamiento del *matching*.

Así, concluimos que este algoritmo no es adecuado, que hay alguna circunstancia que no permite que el *matching* se ejecute de forma correcta. Las hipótesis de cuál pueden ser los motivos son:

[1] A pesar de ser nuestro mapa de prueba una matriz de 1's y 0's, Matlab no la tiene definida como lógica así que debemos darle este formato usando el operador *logical()*.

[2] La tercera subversión, correspondiente a los obstáculos móviles, se ha dejado de lado hasta obtener una versión que responda adecuadamente para las dos primeras subversiones.

- Con la *Occupancy Grid* booleana el *matching* funcionaba correctamente ya que en caso de descuadrarse el mapa, los valores eran directamente 0, pero en el caso probabilístico, se cogen celdas contiguas a la correcta, que pueden tener valores no tan despreciables como es el caso del interior de los obstáculos o las zonas no vistas (valor 50). Así, cuando la celda correcta tiene un valor no muy elevado (cercano al 50), puede ser que con un mapa desplazado haya falsas elecciones de mayor correspondencia al contrarrestarse el valor positivo de celdas no visualizadas/interior de objetos con el valor negativo de celdas con probabilidad baja.
- En la primera iteración, que es donde tenemos datos 100 % fiables, solo se realiza una fase de incrementos/decrementos por lo que el mapa resultante es muy parecido al mapa inicial y no tenemos una buena referencia de partida para el *matching*.[3]

Al observar que el *matching* no funciona, vamos a omitir el análisis referente al error de medida y objetos móviles.

4.2. Versión 8 del Algoritmo de Mapeo

Debido al fracaso de la séptima versión del mapeador, se propone una nueva versión que implementa un nuevo método de comparación entre mapas. Ahora tanto el mapa global como los mapas de prueba serán de tipo probabilístico y el método de comparación consiste en que haya la menor diferencia posible entre las celdas equivalentes de ambos. Es decir que para celda, se calculará la diferencia entre ambos mapas y se tomará el valor absoluto. Posteriormente, se sumarán las diferencias calculadas de cada celda para dar paso al indice de correspondencia inversa[4] del actual mapa de prueba.

$$correspondenciainv = \sum_{j}^{m}\sum_{k}^{m}|MapaGlobal(j,k) - MapaPrueba(j,k)| \qquad (4.1)$$

Donde MapaGlobal y MapaPrueba son matrices mxm.

De esta forma pretendemos incluir en el cálculo no solo aquellas celdas con obstáculos sino todas, incluyendo las correspondientes al interior de los obstáculos y a zonas no visibles. Una vez calculadas todas las correspondencias inversas, se debe tomar el mapa con la menor. Este debe ser en teoría, un mejor *matching* que la versión anterior ya que se consideran tanto las coincidencias de obstáculo como las de no obstáculo, las de incertidumbre, etc.

[3]Recordar que como se explicó en el apartado 3.2.2, el mapa resultante de la primera iteración es una pieza clave para el correcto desempeño del *matching* durante el resto de iteraciones.

[4]Notar que en este caso, la mayor correspondencia se dará cuanto menor sea la diferencia entre las celdas de ambos mapas, siendo 0 si ambos mapas fuesen exactamente iguales.

4.2.1. Implementación

Para la implementación de la función *Mapping_v8* cuyo código se puede consultar en el apéndice B, se ha tenido en cuenta las siguientes consideraciones:

- Para hallar el mapa de prueba en base a modificar el mapa global, se hace uso de la función *CalculateMAP_v3*.
- Para comparar ambos mapas se utiliza un bucle doble que realiza un barrido donde recorre cada posición de las matrices. Al igual que se hizo en anteriormente, se define un parámetro de precisión que determina el incremento de celda en el barrido, es decir que un valor de precisión de 1 significa que compararemos todas las celdas y un valor de 3 que compararemos una de cada tres celdas en cada dimensión de la matriz. Esto se hace por si existe la posibilidad de quitar peso computacional al algoritmo sin disminuir demasiado su desempeño.
- Para comparar el valor mínimo se usa una variable que indica el mínimo valor hasta el momento, antes de empezar se define como NaN para poder almacenar la primera medida.[5]
- El código tiene comentadas dos líneas que muestran por pantalla todos los valores de correspondencia inversa así como el vector de ganadores. Esto se hace así para una posible labor de depuración que se quiera realizar.

4.2.2. Entorno de Simulación

No representa ningún cambio respecto a la versión anterior. La función ha sido llamada *MappingTB_v9* y cuenta con dos subversiones[6] similares a anteriores versiones. Los códigos se pueden consultar en el apéndice B y los resultados obtenidos de las simulaciones en el apéndice A.

4.2.3. Evaluación de Resultados

Los resultados no han sido precisamente buenos, el error es acumulativo con una media alta para todos los casos, lo cual demuestra que el *matching* no está haciendo su trabajo. El desempeño de este es peor aun que en la versión anterior.
Además, el tiempo de ejecución del algoritmo de mapeo ha aumentando considerablemente, 1694.92ms por ejecución, comparado a los 414.68ms de la versión anterior. Estos valores se pueden consultar en las figuras A.58 y A.75. Las hipótesis del fracaso de esta versión son:

- Al igual que la versión anterior, para la primera iteración solo se realiza una operación de incrementos/decrementos por lo que el nuevo mapa se parece mucho al inicial y no tenemos una buena referencia de partida para el *matching*.

[5]Para no tener que poner un valor inicial alto y arriesgarse a que los valores de correspondencia estén por encima de este.

[6]La tercera subversión, correspondiente a los obstáculos móviles, se ha dejado de lado hasta obtener una versión que responda adecuadamente para las dos primeras subversiones.

- Al trabajar con diferencias entre dos mapas muy parecidos (recordar que sobretodo al principio, los valores de todas las celdas son cercanos al 50), al probar con mapas desplazados puede que se incurra en falsas elecciones del mapa más coincidente al contrarrestarse diferencias grandes con diferencias pequeñas.

- En general, los mapas de prueba tienen una operación de incrementos/decrementos de más respecto al mapa global, con lo que incluso sin error y en la pose central, el mapa de prueba no será exactamente igual al global. Esto se había aceptado ya que al ser una característica común a todos los mapas de prueba y afectar a todos por igual, en teoría no debería afectar al grado relativo de coincidencia, pero quizás hay algún aspecto que estemos pasando por alto y sí que afecte al *matching*.

De esta forma, descartamos el algoritmo por no responder adecuadamente a las especificaciones. Al no pasar las pruebas correspondientes al error de odometría, se omite el análisis correspondiente al error de medida y obstáculos móviles.

4.3. Versión 9 del Algoritmo de Mapeo

Después de haber obtenido malos resultados en las dos últimas versiones, debemos decidir el siguiente paso, si probar con otro método de matching o intentar modificar los ya vistos para mejorar su respuesta. De mejorar los métodos vistos, la opción que parece la más acertada es mejorar el método visto en la séptima versión por dos motivos, porque los resultados que presentó fueron mejores que el método de la octava versión y también porque su tiempo de ejecución es mucho menor. Así pues, será este método el que se intentará corregir por medio de algunas modificaciones referentes a las hipótesis de su fracaso vistas en el apartado 4.1.

La primera modificación que se ha probado es dar mayor peso a la primera iteración para obtener una mejora referencia de partida, para ello se realiza la operación de incrementos/decrementos para un número acordado de veces que se ha fijado en 5. Esta modificación ha tenido buenos resultados, aumentando el numero de aciertos del *matching*, por lo que se deja implementada.

La siguiente modificación consistió en desechar todas las celdas menores de un umbral, es decir, considerar para la media de la coincidencia únicamente las celdas con un valor por encima de este, que se fijó en 55. Esto se hizo con el objetivo de no tener en cuenta las celdas con incertidumbre (valores próximos a 50) como son las correspondientes al interior de obstáculos o a zonas no vistas. Por razones desconocidas, esta modificación no ha tenido buenos resultados, observando con ella un *mathcing* caótico. No se conserva esta modificación.

Se ha comprobado de igual forma la compatibilidad entre las funciones que calculan la *Occupancy Grid* booleana y la probabilística ya que cabía la posibilidad de que algún error hubiese ocasionado un desfase entre los dos tipos de mapa o cualquier otra incompatibilidad de ese estilo. Se comprueba que ambas funciones son totalmente compatibles.

Por último, en un desesperado intento por mejorar el rendimiento, se ha intentado emular lo más posible el *matching* que vimos en la quinta versión del mapeador (apartado 3.2.2). Para ello, se ha umbralizado el mapa global probabilístico para obtener un mapa booleano dependiente de este y así poder realizar la comparación tal como lo hacíamos en la versión *Mapping_v5*. Esta argucia ha arrojado resultados ligeramente mejores pero no convincentes aun. Para mejorar la comparación booleana, se ha realizado una submodificación que considera no solo los 1's en común sino también los 0's, para ello es necesario umbralizar el mapa probabilístico tanto por

arriba como por debajo. Es decir, considerar 0's los valores que estén por debajo de un primer umbral y 1's los valores que estén por encima de un segundo. Así, se han conseguido resultados aun ligeramente mejores.

Después de hacer una larga y tediosa labor de depuración, intentando comparar con el *matching* de la versión 5, se ha determinado como última medida disminuir el valor del decremento respectivo al camino del haz de luz hasta el objeto de -3 a -1. Los resultados han mejorado notablemente, para sorpresa de todos.

Cabe destacar que el rendimiento de todas las modificaciones mencionadas ha sido probado únicamente bajo condiciones de error de odometría y no sabemos como vaya a responder ante el error de medida.

4.3.1. Implementación

Para la implementación de las modificaciones mencionadas, se han tenido en cuenta las siguientes consideraciones:

- Para la dramatización de la primera medida (darle más peso), se ha hecho uso de un bucle que llama a la función *CalculateMAP_v3* un número determinado de veces. Este número se ha definido como el parámetro interno de la función *dramalevel* y se ha fijado por defecto en 5.
- Para la umbralización del mapa probabilístico se ha hecho uso de los operadores ¡y ¿que dan como resultado matrices lógicas que luego se combinan adecuadamente con operadores AND y NOT para conseguir lo deseado. Los umbrales superior e inferior se definen como parámetros internos llamados *umbSup* y *umbInf* respectivamente. Por defecto se fijan en 40 y 60.
- Dentro de *CalculateMAP_v3*, se ha modificado el valor del parámetro interno *DecAnteObjeto* de -3 a -1.
- El resto permanece similar a *Mapping_v7*.

La función implementada ha sido llamada *Mapping_v9* y el código se puede consultar en el apéndice B.

4.3.2. Entorno de Simulación

No representa ningún cambio respecto a la versión anterior. La función ha sido llamada *MappingTB_v10* y cuenta con tres subversiones similares a anteriores versiones. Los códigos se pueden consultar en el apéndice B y los resultados obtenidos de las simulaciones en el apéndice A.

4.3.3. Evaluación de Resultados

Al estar en la versión final, que implementa ambas correcciones de ruido, vamos a hacer el análisis un poco más profundo. En general observamos un comportamiento bueno, errores no acumulativos que si bien en ocasiones dan un cambio muy brusco, mayormente se mantienen constantes.

Para las simulaciones referentes a la primera subversión *MappingTB_v10_1* podemos observar en la figura A.88 que para errores medios (odometría 0.5, medida 0.1) se obtiene un comportamiento bastante bueno, que si bien el error sube un poco en las primeras iteraciones (debido al mapa inicial de referencia baja) luego se mantiene casi constante con ligeras subidas en los puntos que se curva la trayectoria y se cambia bruscamente el rango de visión, lo cual es un problema del entorno de simulación con obstáculos muy angulosos y giros fuertes, probablemente en la realidad tendríamos un comportamiento más lineal. Para la misma subversión pero con errores mayores (odometría 0.8, distancia 0.1) observamos que sigue habiendo un comportamiento correcto, el mapa aun es representativo y el error cuadrático medio es relativamente bajo, en la figura A.95 podemos observar lo mismo que se expuso anteriormente, que a pesar de haber ocasiones en que el error sube drásticamente, mayormente intenta mantenerse constante.

La segunda subversión representa un poco mejor la realidad, ya que trabaja con velocidades angulares mas realistas para un robot de las dimensiones consideradas. Por ello, y como ya se explicó, se debe trabajar con errores de odometría menores para conservar la relación de ruido en dicha señal. Respecto a las simulaciones, observamos para errores bajos, cercanos a los que se tendrían en la realidad (odometría 0.16, distancia 0.05) un comportamiento casi perfecto con mapas muy representativos, un error cuadrático medio bajo y en la figura A.99 observamos que el error se mantiene bajo y constante hasta el punto en el que el robot cambia bruscamente de vista, se queda sin referencia y el error se dispara (aproximadamente en la iteración 50), pero observamos que luego de la subida, se mantiene constante nuevamente y de volver a tener visión de zonas conocidas, es muy probable que se corrija el error acumulado como veremos en la tercera subversión. Para errores más grandes (0.3 odometría y 0.1 de distancia) observamos un comportamiento muy bueno hasta cierto punto, donde el robot pierde referencia (por cambios de vista y/o errores aleatorios muy grandes) y tiene subidas en el error como podemos apreciar en la figura A.103. Aun así, el comportamiento es bueno, ya que la gráfica muestra que hasta aproximadamente la iteración 50, se ha logrado mantener constante el error, aun después del crecimiento presentado en la iteración 35.

La tercera subversión es la mas realista de las tres, ya que implementa la base de tiempos y velocidades de la segunda, tiene un recorrido que permite evaluar el comportamiento al pasar nuevamente por una zona conocida y además incluye obstáculos móviles de tal forma que se puede probar con errores de magnitud real, el comportamiento ante todas las situaciones planteadas en este documento. Para errores bajos observamos un comportamiento muy bueno, con mapas representativos, error cuadrático medio relativamente bajo, los obstáculos móviles no se muestran en el mapa y presenta evolución del error de pose interesante como podemos observar en la figura A.107, en esta apreciamos que el error se intenta mantener constante hasta la iteración 43, donde el robot llega al extremo del recinto y se queda casi totalmente sin referencias para el *matching* por lo que el error tiene una pendiente elevada, pero vemos que tan pronto gira y vuelve a tener referencias, el error empieza a disminuir progresivamente mientras mas avanza y mejor referencia tiene de las zonas ya vistas de tal forma que termina corrigiendo el error casi por completo. En la figura A.113 podemos comprobar la robustez de esta simulación. Para errores mayores (0.2 de odometría y 0.1 de distancia) observamos un comportamiento bueno pero

menos perfecto que el anterior, el mapa sigue siendo representativo, con pocas celdas positivas fuera de los obstáculos. Sin embargo, el error cuadrático medio ya no es tan bajo, principalmente debido al descuadre de la pose en las primeras iteraciones como podemos observar en la figura A.112, debido al error de distancia le ha costado coger la referencia al principio, por lo que el error ha aumentado hasta aproximadamente 0.3, donde parece que se ha estabilizado y ha empezado a mantener constante el error. Luego, al volver a zonas conocidas el error ha disminuido al corregirse por la gran cantidad de referencias, pero no ha llegado a valores realmente bajos, quizás por falta de tiempo/iteraciones. Así, podríamos decir que para esta subversión, esta simulación responde al límite de estabilidad de nuestro algoritmo de mapeo, para errores gaussianos de desviación típica 0.2 para el caso de la odometría y 0.1 para el caso de las distancias. Respecto al tiempo de ejecución, podemos observar en la figura A.108 que es de 5.785s para 90 iteraciones, lo que da un total de 64.28ms de tiempo de ejecución individual para el algoritmo de mapeo, un tiempo más que suficiente para poder ejecutarse en tiempo real.

Conclusions

We have developed several mapping algorithms which progressively grow in complexity and functionality, starting by the simplest one and introducing new features as we moved in new versions. Also we have developed a simulation environment in order to evaluate the performance of each of those. From the simulation results we can conclude:

- The mapping problem is a complex field where things can get complicated by several reasons, principally by the measurement's errors. For this reason, it has to be found a mapping method that fits onto the particular case where we are so it can work correctly and efficiently solving the particular specifications.
- We started working with a continuous coordinates map but it had to be replaced by an discrete coordinates one, known as *Occupancy Grid*. This solution solves many map-handling problems since it's far easier to work with a bounded number of data than with continuous values with almost infinitesimal variations.
- The *boolean Occupancy Grid* works fine but has an important defect which is to be boolean since in real situations we will never have full certainty of a value with a only measure but we would need several of them. Also, the *boolean Occupancy Grid* has the defect of remembering a value forever, which is unwanted insomuch as we will sometimes have false measures that should be eventually discarded.
- The tested method for solving the odometry error issue has been the *map matching* and it has demonstrated to work correctly under certain conditions. These are having a high level of accuracy in the sensing system, not too big errors in the odometry measurement, always having a reference sight of the already seen objects with a soft movements based trajectory, a well-known starting position and the main, having a suitable model of the odometry error so we can know his typical deviation. Even under this conditions, the method fails sometimes due to unpredictable situations so it's not recommended to use this method in cases where is strictly needed a high reliability operation such as autonomous vehicles in public spaces.
- The tested method for solving the distance error issue has been the implementation of a *probabilistic Occupancy Grid* which has had good results in the simulations. This method resolves the problem of false measures and it can be useful in many other aspects such as forgetting moving objects or forgetting a false measure because of a pose error (as long as this error has been corrected).
- Merging the two methods has brought new issues since the probabilistic map makes the matching operation more sensitive. In this case, it fails more often and both of the errors must be under a certain limit so it can operate with an acceptable level of reliability.

Therefore, it's recommended to use both methods merged only when necessary, according to the particular case needs.

Other conclusions:

- When coding, it's always important to make your code easily customizable so if somewhen you need to make some changes, there's no need to change almost everything. For achieving this, it's important to think about possible modifications that could be done in the future and make an easy way to them. Some technics of this ideology are: to parameterize the most relevant variables so it can be easily modified, to comment on all relevant lines, to indent the lines according to his hierarchy, etc.
- Sometimes it's useful to create your own algorithms istead of using the third party's because yours will do exactly what you need, not more not less, and that can make the algorithm far more efficient than a general purpose one.

Future Work

In order to continue the exposed work, some suggestions are made:

- Test new methods of correcting the odometry error, such as more complex matchings with a more complex sensing system.
- Implement a loop feedback in the localizaton process.
- Implement the complete SLAM with a route planner and obstacle avoidance.
- Test the algorithms on a real environment with all the sensors and motors controllers.
- Evaluate the possibility of using another kind of sensors instead of the laser telemeter.

Apéndices

Apéndice A

Resultados de las Simulaciones

A continuación se exponen los resultados obtenidos para cada entorno de simulación. Para cada simulación se detalla la línea de código correspondiente a la llamada a la función desde la consola de *Matlab*. El formato general es:

MappingTB_v10[1]_3[2](0.16[3],0.05[4])

Así mismo se presentan a continuación de cada simulación identificada por su respectiva llamada, el resultado gráfico de la simulación, el/los mapas resultantes, ampliaciones de los mismos, tablas de errores de pose, gráficas de errores de pose, resultados del análisis de desempeño temporal, entre otros.

Respecto a los resultados gráficos, *Matlab* realiza una simulación de aproximadamente 80 iteraciones y el gráfico cambia para cada iteración. Por razones lógicas no se pueden mostrar aquí las 80 gráficas, con lo que expone una muestra representativa de estas de forma que se pueda apreciar el comportamiento de la simulación sin lugar a malentendidos.

Respecto a las tablas y gráficas de la evolución del error en la pose, se ha considerado el error cuadrático tridimensional, que responde a la siguiente fórmula:

$$Error_{pose} = \sqrt{(x - x_p)^2 + (y - y_p)^2 + (\theta - \theta_p)^2} \tag{A.1}$$

La explicación de cada versión y subversión se puede consultar en el respectivo apartado de los capítulos 2, 3 y 4, así como la evaluación y discusión de los resultados obtenidos.

Para las dos últimas tablas de numeración A.23 y A.24, se han omitido las filas correspondientes a las tres últimas iteraciones por razones estéticas. Aun así, los valores omitidos constan en la gráfica correspondiente y han sido considerados para el cálculo del *error cuadrático medio*.

[1] Versión del entorno de simulación
[2] Subversión del entorno de simulación
[3] Desviación típica del error de odometría
[4] Desviación típica del error de distancia

A.1. MappingTB_v1

```
1 MappingTB_v1()
```

Figura A.1: Resultados de *MappingTB_v1*, obstáculos en amarillo, mapa en rojo y robot en azul

A.2. MappingTB_v2_1

```
1 MappingTB_v2_2()
```

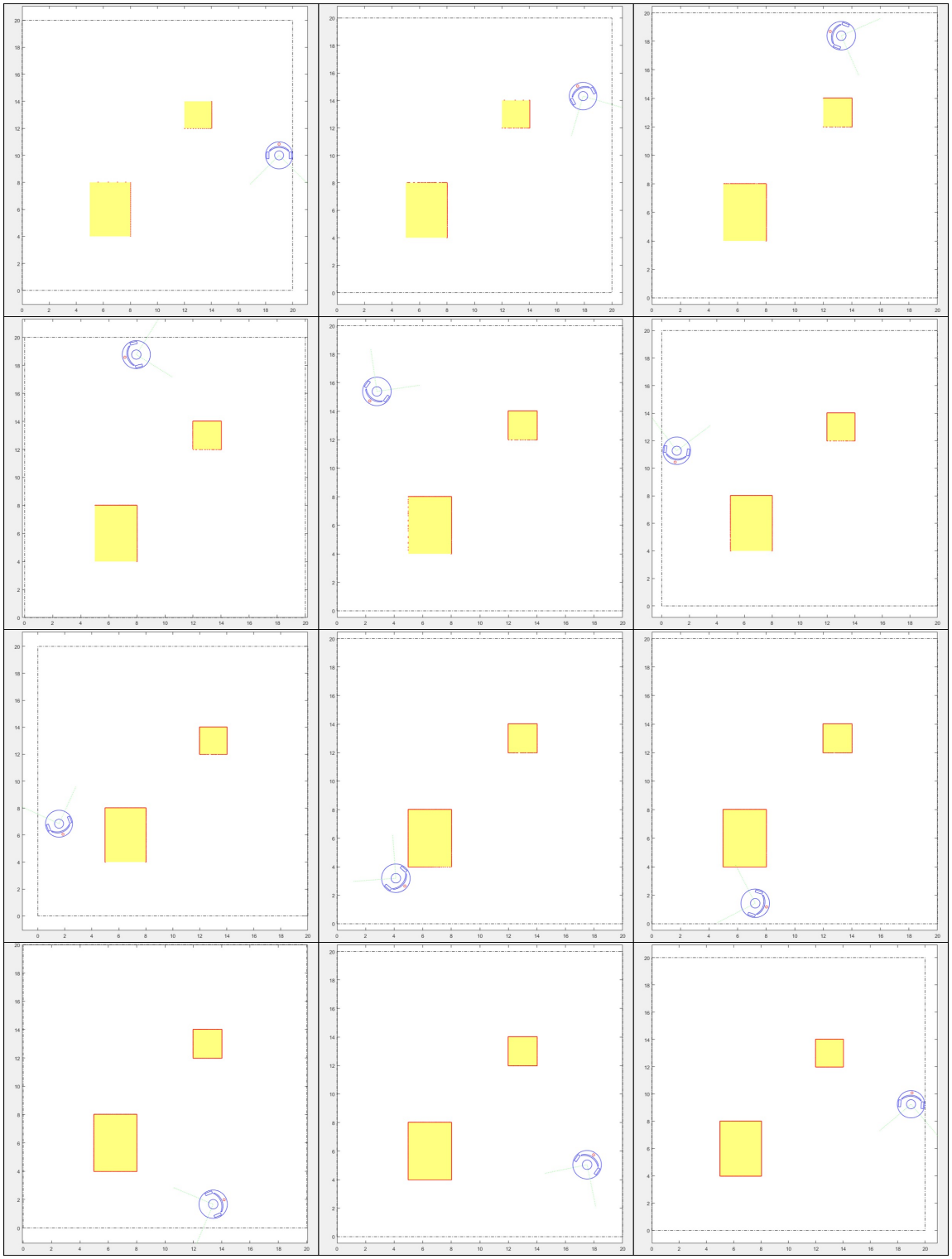


Figura A.2: Resultados de *MappingTB_v2_1*, obstáculos en amarillo, mapa en rojo y robot en azul

A.3. MappingTB_v2_2

```
1 MappingTB_v2_2()
```

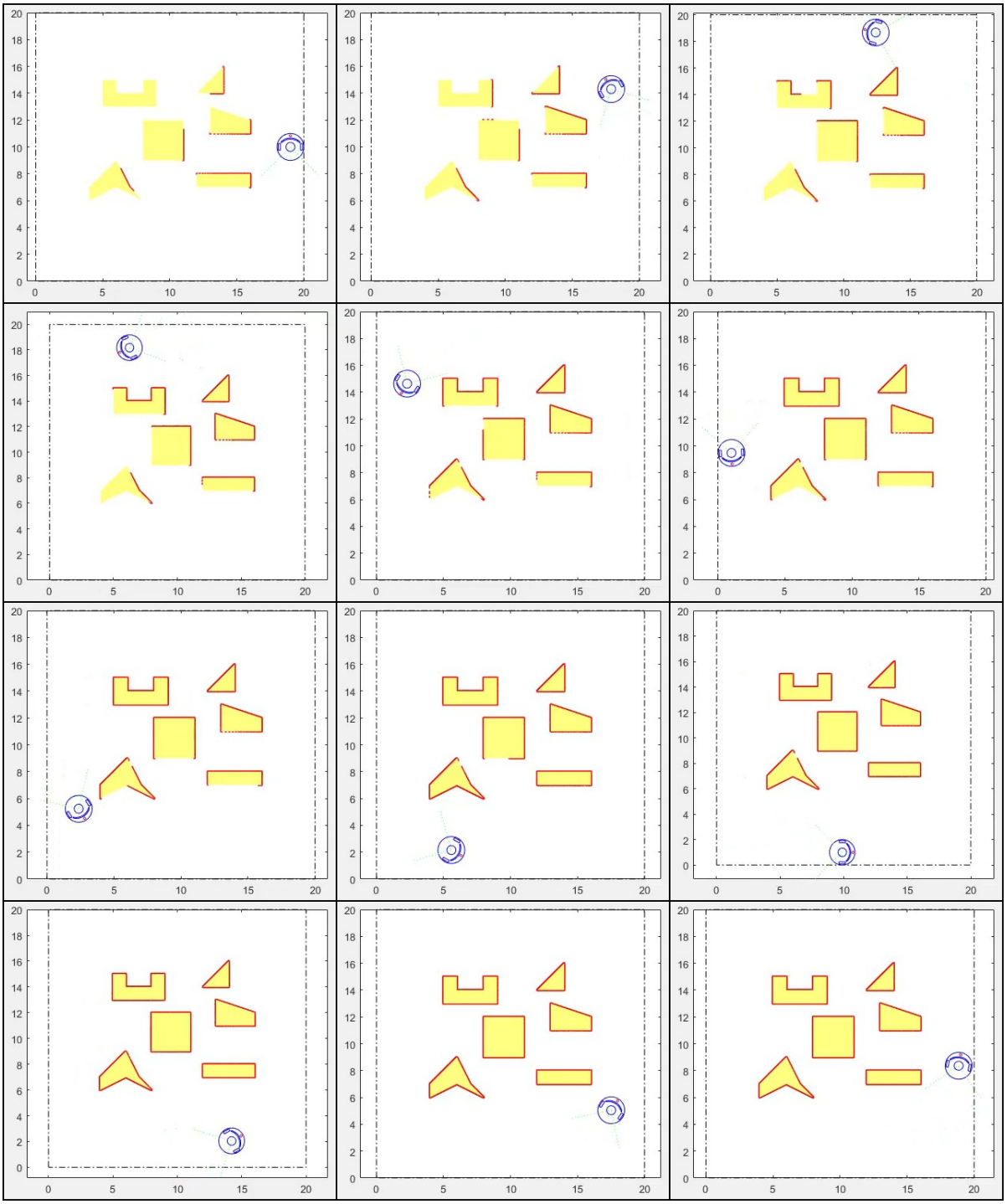


Figura A.3: Resultados de *MappingTB_v2_2*, obstáculos en amarillo, mapa en rojo y robot en azul

A.4. MappingTB_v3_1

```
1 MappingTB_v3_1()
```

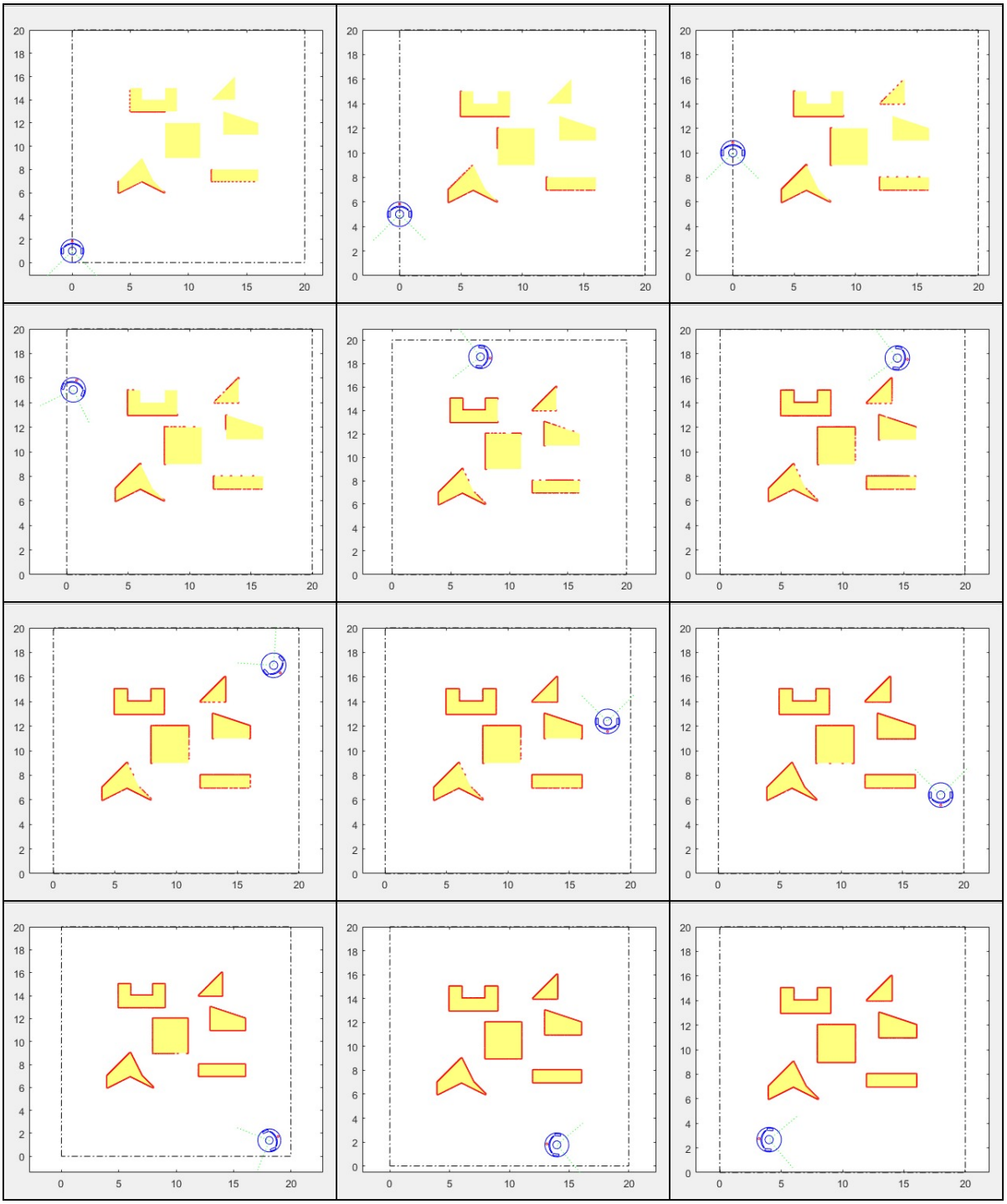


Figura A.4: Resultados de *MappingTB_v3_1*, obstáculos en amarillo, mapa en rojo y robot en azul

A.5. MappingTB_v4_1

```
1 MappingTB_v4_1(0,0.1)
```

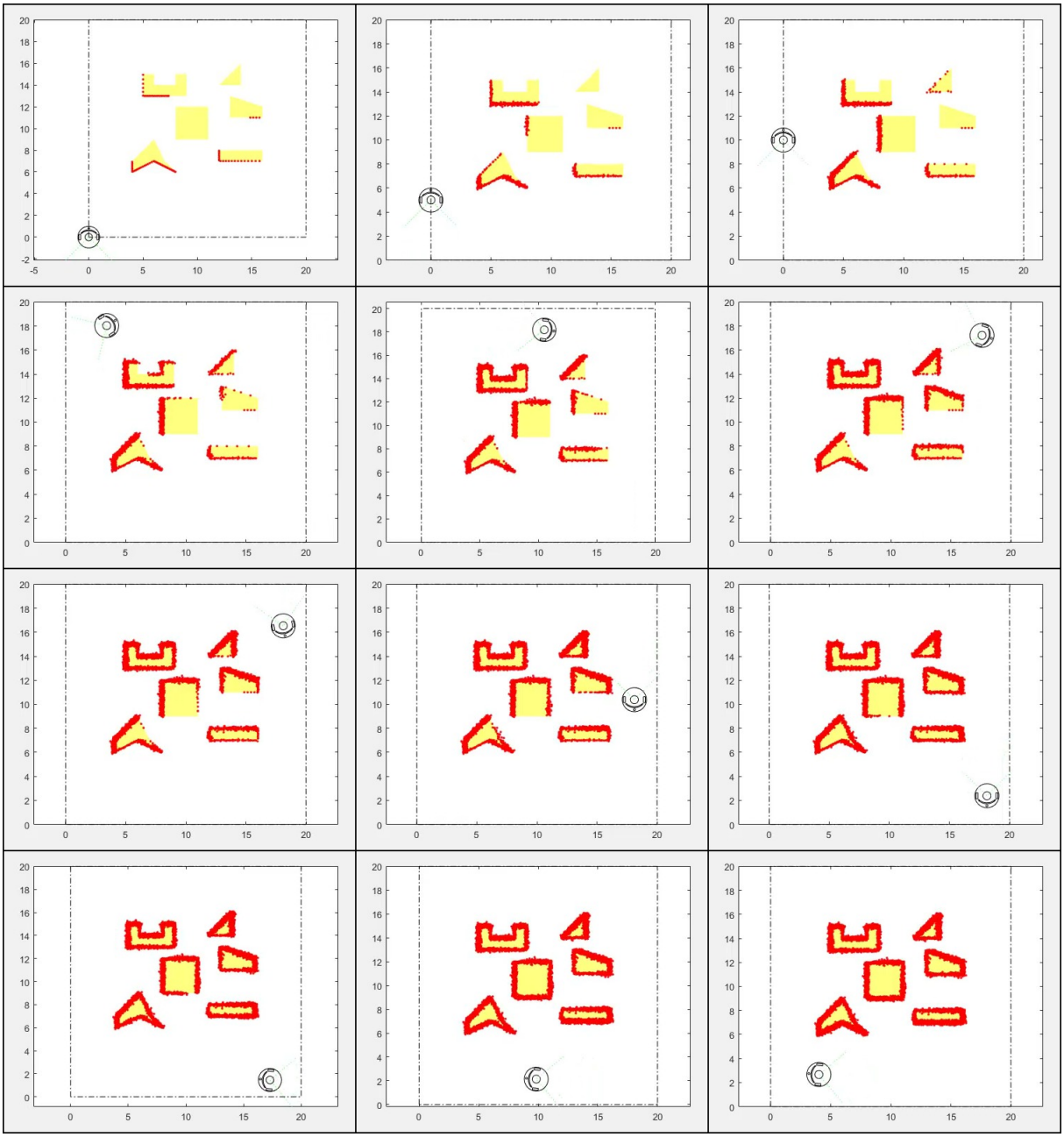


Figura A.5: Resultado de *MappingTB_v4* con error de medida. Obstáculos en amarillo, mapa en rojo, pose real en verde y pose calculada en negro.

```
1 MappingTB_v4_1(0.8,0)
```

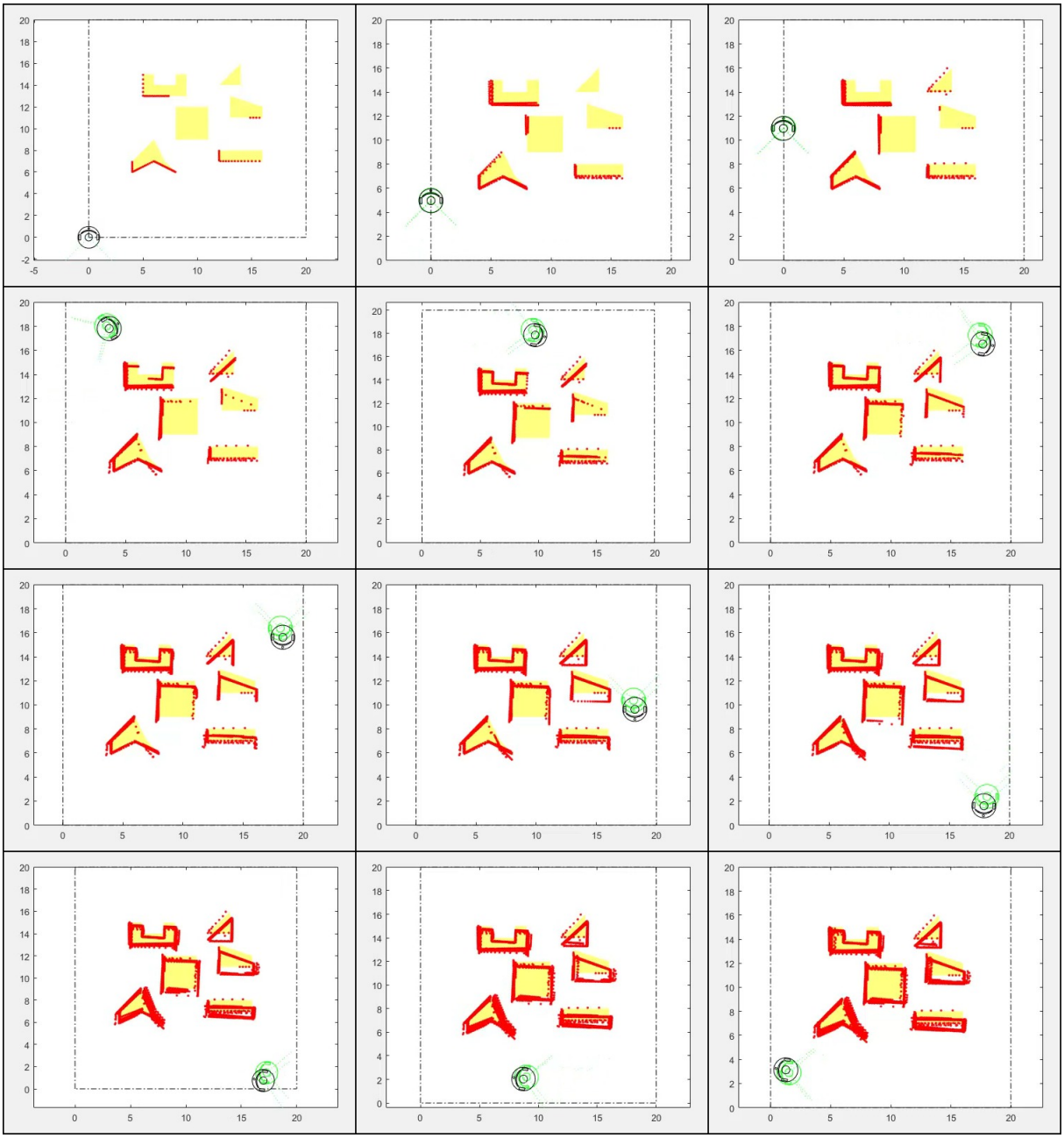


Figura A.6: Resultado de *MappingTB_v4* con error de odometría. Obstáculos en amarillo, mapa en rojo, pose real en verde y pose calculada en negro.

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0	1	0	0.0169
3	0	2	0	0.0221
4	0	3	0	0.029
5	0	4	0	0.0285
6	0	5	0	0.0298
7	0	6	0	0.0398
8	0	7	0	0.0553
9	0	8	0	0.0749
10	0	9	0	0.0859
11	0	10	0	0.0938
12	0	11	0	0.1126
13	0	12	0	0.1385
14	0	13	0	0.1896
15	0.179	14.04	-0.17	0.2182
16	0.532	15.04	-0.34	0.2405
17	1.049	15.96	-0.51	0.2853
18	1.714	16.79	-0.68	0.3428
19	2.509	17.49	-0.85	0.3853
20	3.411	18.04	-1.02	0.418
21	4.394	18.43	-1.19	0.4297
22	5.428	18.66	-1.36	0.4505
23	6.486	18.7	-1.53	0.4609
24	7.535	18.56	-1.7	0.4634
25	8.527	18.43	-1.7	0.4572
26	9.519	18.3	-1.7	0.4477
27	10.51	18.18	-1.7	0.4615
28	11.5	18.05	-1.7	0.4668
29	12.49	17.92	-1.7	0.451
30	13.48	17.79	-1.7	0.4475
31	14.48	17.66	-1.7	0.4437
32	15.47	17.53	-1.7	0.4578
33	16.46	17.4	-1.7	0.4566
34	17.45	17.27	-1.7	0.4451
35	17.59	17.23	-1.88	0.452
36	17.73	17.16	-2.06	0.4478
37	17.84	17.06	-2.24	0.4518
38	17.94	16.95	-2.42	0.447
39	18.02	16.82	-2.6	0.4437
40	18.07	16.68	-2.78	0.4539
41	18.1	16.54	-2.96	0.4511
42	18.1	16.39	-3.14	0.4581
43	18.1	15.39	-3.14	0.4717
44	18.11	14.39	-3.14	0.4746
45	18.11	13.39	-3.14	0.4838

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
46	18.11	12.39	-3.14	0.4874
47	18.11	11.39	-3.14	0.4799
48	18.11	10.39	-3.14	0.4661
49	18.11	9.385	-3.14	0.45
50	18.11	8.385	-3.14	0.4303
51	18.12	7.385	-3.14	0.4277
52	18.12	6.385	-3.14	0.4117
53	18.12	5.385	-3.14	0.3794
54	18.12	4.385	-3.14	0.3476
55	18.12	3.385	-3.14	0.3363
56	18.12	2.385	-3.14	0.3202
57	18.13	1.385	-3.14	0.301
58	18.13	1.385	-2.48	0.2848
59	18.13	1.385	-1.82	0.2917
60	18.13	1.385	-1.16	0.2962
61	18.13	1.385	-0.5	0.2954
62	18.13	1.385	0.16	0.2997
63	18.13	1.385	0.82	0.2922
64	18.13	1.385	1.48	0.2931
65	17.3	1.46	1.48	0.2608
66	16.47	1.536	1.48	0.2388
67	15.64	1.611	1.48	0.2135
68	14.81	1.687	1.48	0.1804
69	13.98	1.763	1.48	0.1896
70	13.15	1.838	1.48	0.2028
71	12.32	1.914	1.48	0.212
72	11.49	1.989	1.48	0.2302
73	10.66	2.065	1.48	0.2673
74	9.827	2.14	1.48	0.3109
75	8.997	2.216	1.48	0.3623
76	8.167	2.291	1.48	0.414
77	7.337	2.367	1.48	0.4596
78	6.507	2.443	1.48	0.5263
79	5.677	2.518	1.48	0.5925
80	4.847	2.594	1.48	0.6737
81	4.017	2.669	1.48	0.746
82	3.188	2.745	1.48	0.8098
83	2.358	2.82	1.48	0.8768
84	1.528	2.896	1.48	0.9391
Error cuadrático medio				0.3545

Tabla A.1: Error de pose en *MappingTB_v4*(0.8,0)

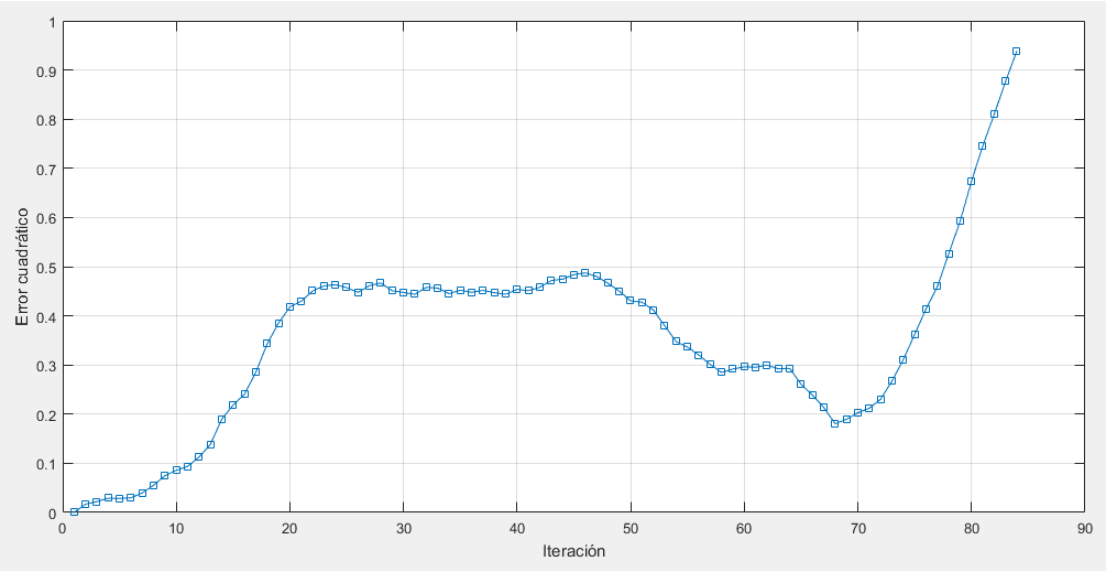


Figura A.7: Gráfica del error de pose en *MappingTB_v4_1*(0.8,0)

A.6. MappingTB_v5_1

```
1 MappingTB_v5_1(0,0)
```

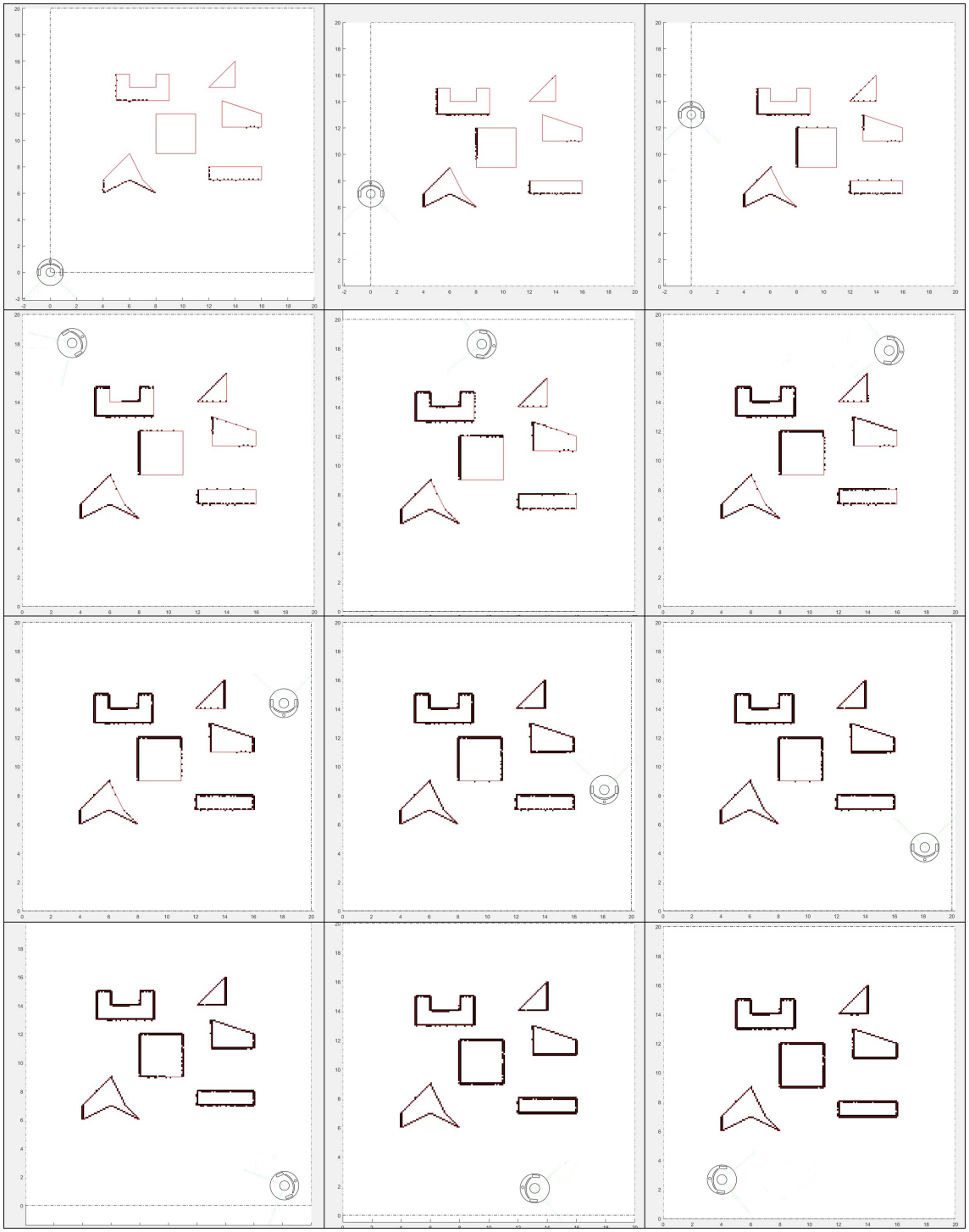


Figura A.8: Resultado de *MappingTB_v5_1* sin errores. Obstáculos en amarillo, mapa en rojo, pose real en verde y pose calculada en negro.

```
1 MappingTB_v5_1(0.8,0)
```

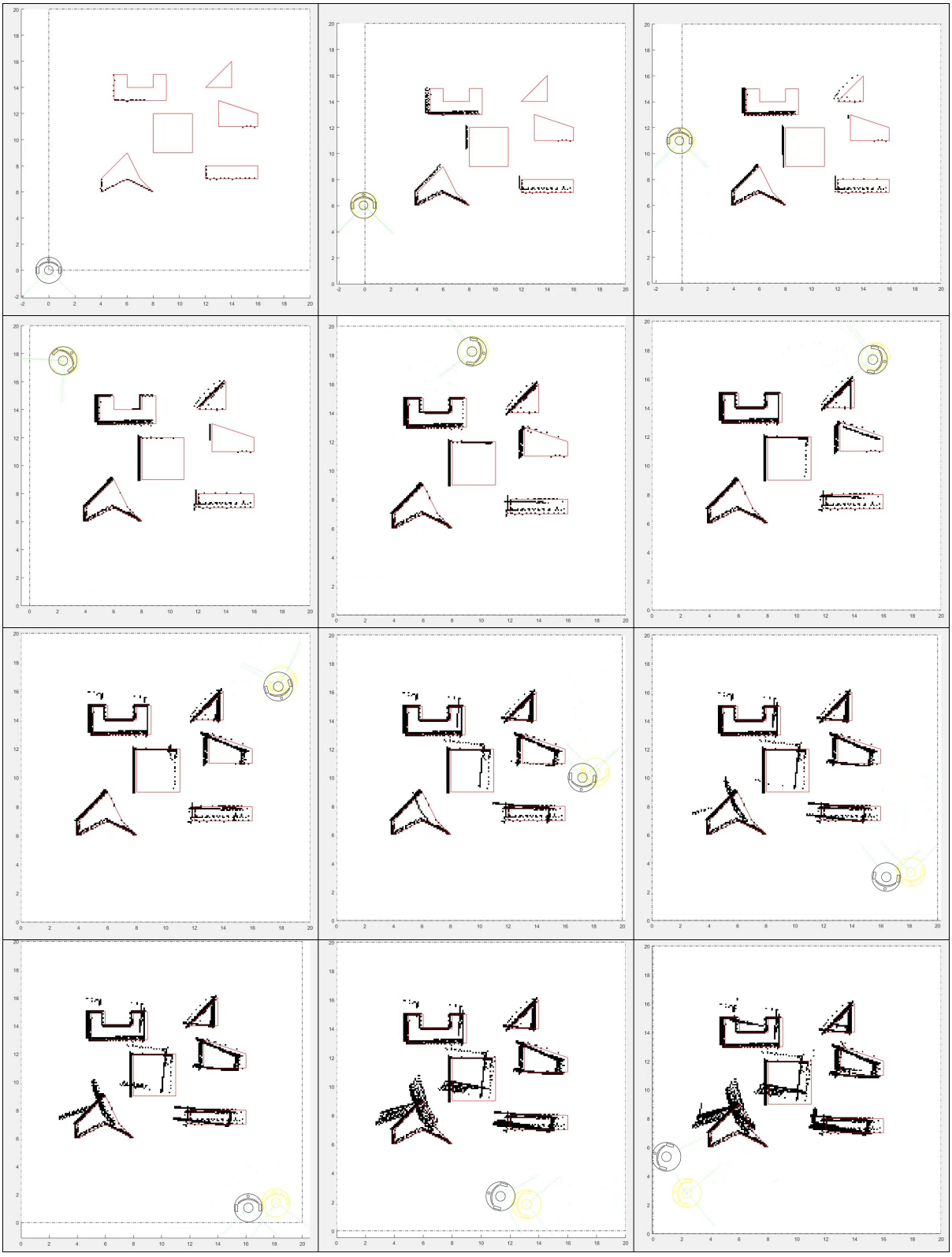


Figura A.9: Resultado de *MappingTB_v5_1* con error de odometría. Obstáculos en rojo, mapa en negro, pose real en amarillo y pose calculada en negro.

```
1 MappingTB_v5_1(0,0.1)
```

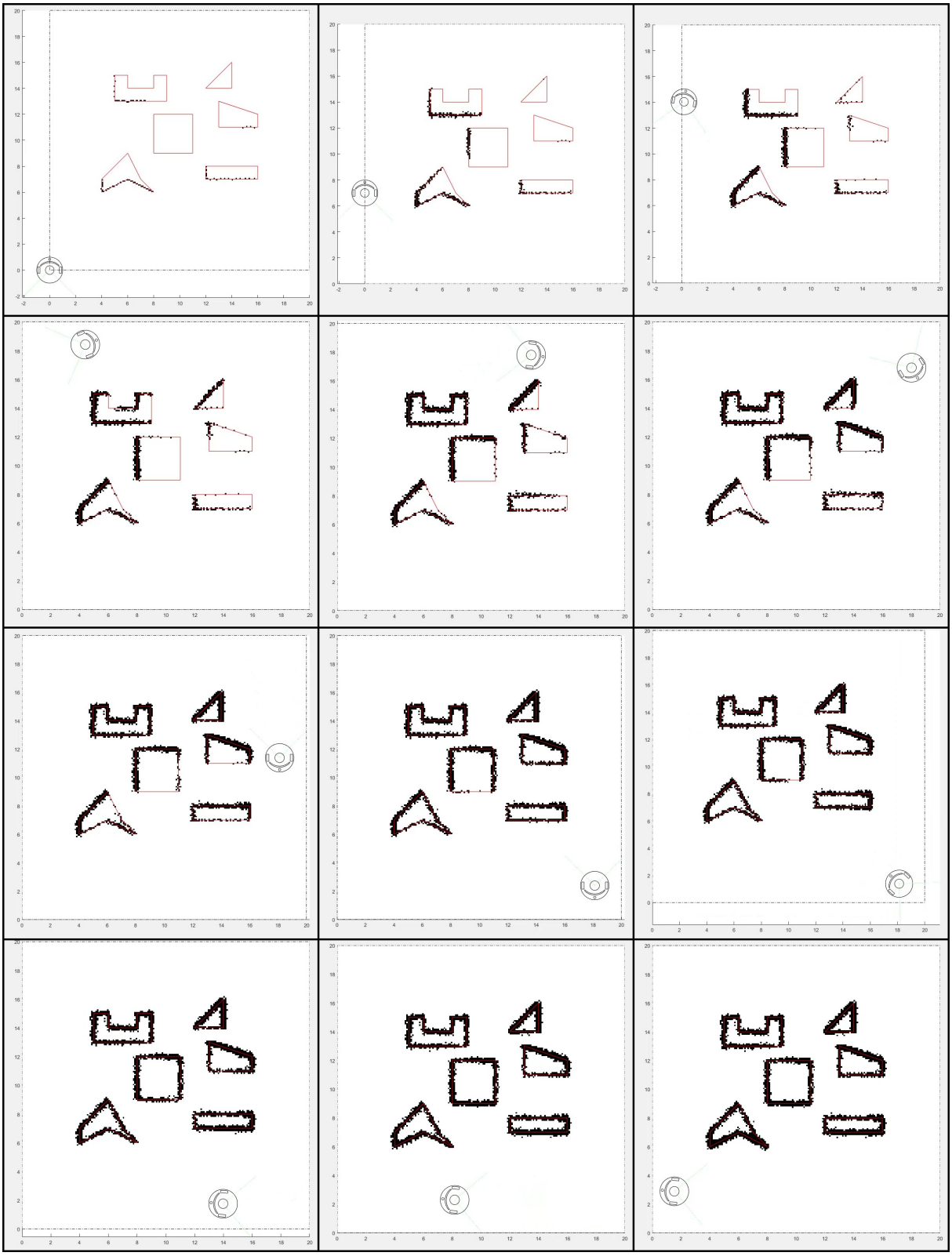


Figura A.10: Resultado de *MappingTB_v5_1* con error de medida. Obstáculos en rojo, mapa en negro, pose real en amarillo y pose calculada en negro.

A.7. MappingTB_v5_2

```
1 MappingTB_v5_2(0.16,0)
```

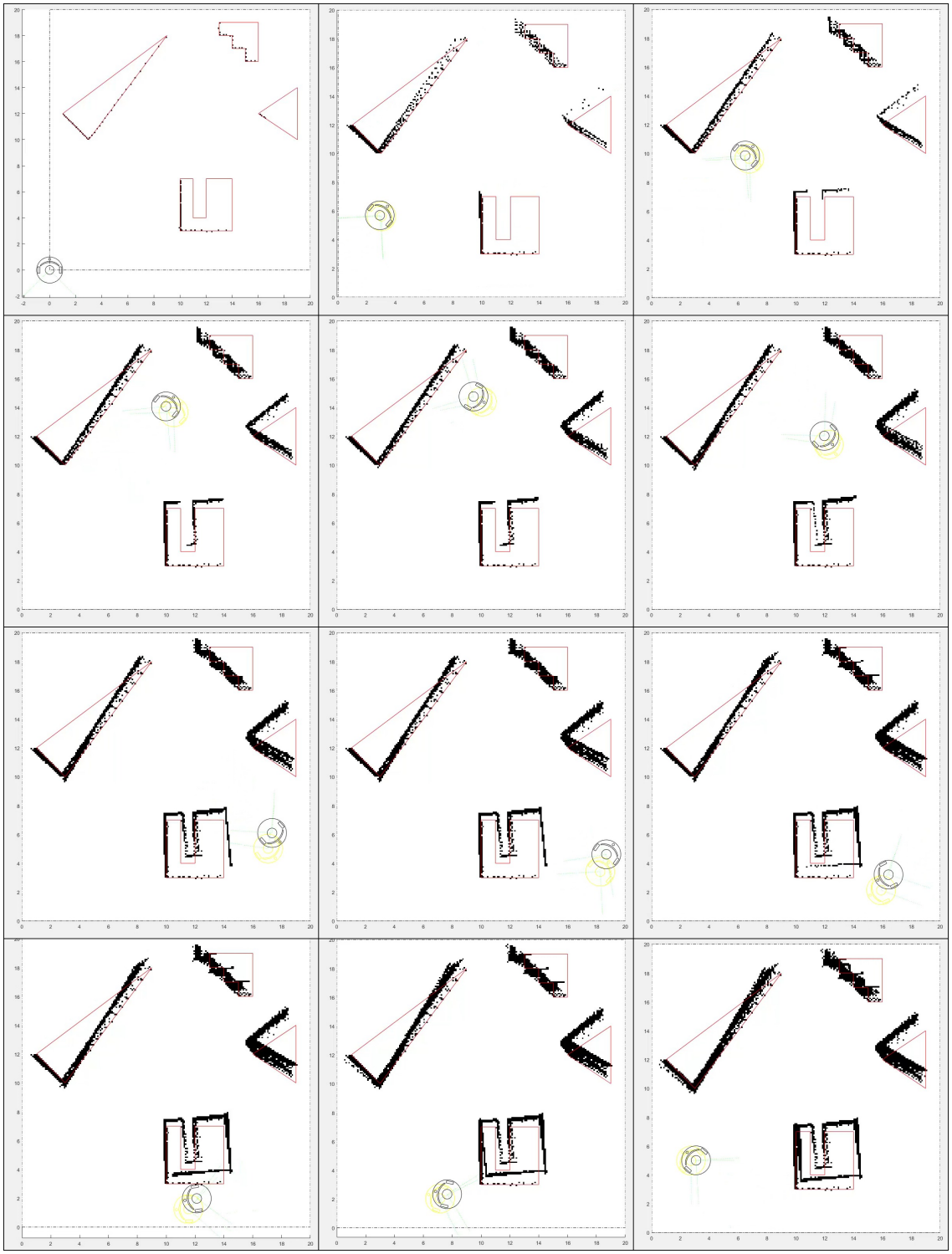


Figura A.11: Resultado de *MappingTB_v5_2* con error de odometría. Obstáculos en rojo, mapa en negro, pose real en amarillo y pose calculada en negro.

A.8. MappingTB_v6_1

```
1 MappingTB_v6_1(0.5,0)
```

Figura A.12: Resultado de *MappingTB_v6_1* con error de odometría 0,5. Obstáculos en rojo, mapa en negro, pose real en amarillo y pose calculada en negro.

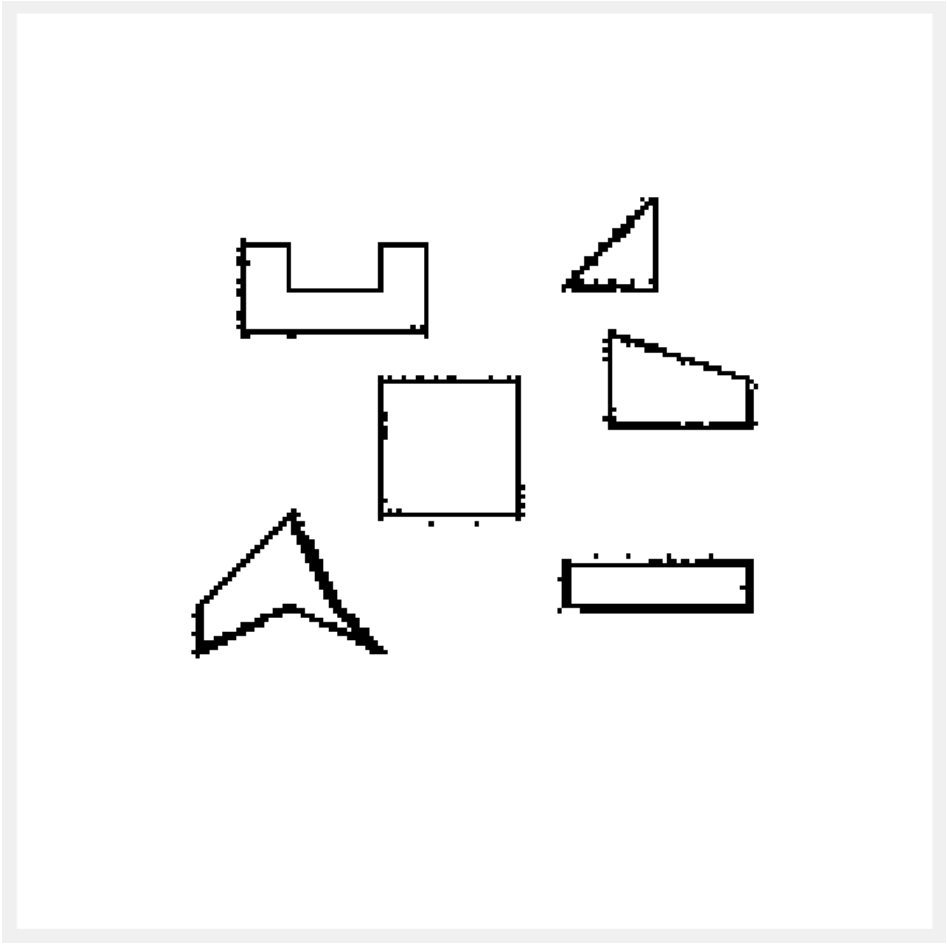


Figura A.13: Mapa resultante de *MappingTB_v6_1* con error de odometría 0,5

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0	1	0	0.0193
3	0	2	0	0.0259
4	0	3	0	0.022
5	0	4	0	0.0281
6	0	5	0	0.0384
7	0	6	0	0.0413
8	0	7	0	0.0434
9	0	8	0	0.0362
10	0	9	0	0.0311
11	0	10	0	0.0304
12	0	11	0	0.0294
13	0	12	0	0.0363
14	0	13	0	0.038
15	0.179	14.04	-0.17	0.0539
16	0.532	15.04	-0.34	0.0584
17	1.049	15.96	-0.51	0.0804

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
18	1.714	16.79	-0.68	0.0497
19	2.509	17.49	-0.85	0.0541
20	3.411	18.04	-1.02	0.066
21	4.394	18.43	-1.19	0.055
22	5.428	18.66	-1.36	0.0509
23	6.486	18.7	-1.53	0.0543
24	7.535	18.56	-1.7	0.0663
25	8.527	18.43	-1.7	0.0634
26	9.519	18.3	-1.7	0.0557
27	10.51	18.18	-1.7	0.069
28	11.5	18.05	-1.7	0.0707
29	12.49	17.92	-1.7	0.072
30	13.48	17.79	-1.7	0.0615
31	14.48	17.66	-1.7	0.0656
32	15.47	17.53	-1.7	0.0518
33	16.46	17.4	-1.7	0.0543
34	17.45	17.27	-1.7	0.0609
35	17.59	17.23	-1.88	0.0459
36	17.73	17.16	-2.06	0.0417
37	17.84	17.06	-2.24	0.0582
38	17.94	16.95	-2.42	0.0623
39	18.02	16.82	-2.6	0.0658
40	18.07	16.68	-2.78	0.0626
41	18.1	16.54	-2.96	0.0577
42	18.1	16.39	-3.14	0.0571
43	18.1	15.39	-3.14	0.0451
44	18.11	14.39	-3.14	0.052
45	18.11	13.39	-3.14	0.0429
46	18.11	12.39	-3.14	0.0519
47	18.11	11.39	-3.14	0.0557
48	18.11	10.39	-3.14	0.0515
49	18.11	9.385	-3.14	0.0448
50	18.11	8.385	-3.14	0.04
51	18.12	7.385	-3.14	0.0234
52	18.12	6.385	-3.14	0.0054
53	18.12	5.385	-3.14	0.0084
54	18.12	4.385	-3.14	0.0169
55	18.12	3.385	-3.14	0.0108
56	18.12	2.385	-3.14	0.0255
57	18.13	1.385	-3.14	0.04
58	18.13	1.385	-2.48	0.0423
59	18.13	1.385	-1.82	0.0388
60	18.13	1.385	-1.16	0.0377
61	18.13	1.385	-0.5	0.0397
62	18.13	1.385	0.16	0.0319

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
63	18.13	1.385	0.82	0.0418
64	18.13	1.385	1.48	0.0396
65	17.3	1.46	1.48	0.0571
66	16.47	1.536	1.48	0.0428
67	15.64	1.611	1.48	0.0309
68	14.81	1.687	1.48	0.0259
69	13.98	1.763	1.48	0.0353
70	13.15	1.838	1.48	0.0493
71	12.32	1.914	1.48	0.0488
72	11.49	1.989	1.48	0.0636
73	10.66	2.065	1.48	0.0586
74	9.827	2.14	1.48	0.0668
75	8.997	2.216	1.48	0.0733
76	8.167	2.291	1.48	0.0675
77	7.337	2.367	1.48	0.0768
78	6.507	2.443	1.48	0.0707
79	5.677	2.518	1.48	0.066
80	4.847	2.594	1.48	0.0658
81	4.017	2.669	1.48	0.0656
82	3.188	2.745	1.48	0.0498
83	2.358	2.82	1.48	0.0495
84	1.528	2.896	1.48	0.0354
Error cuadrático medio				0.04734

Tabla A.2: Error de pose en *MappingTB_v6_1*(0.5,0)

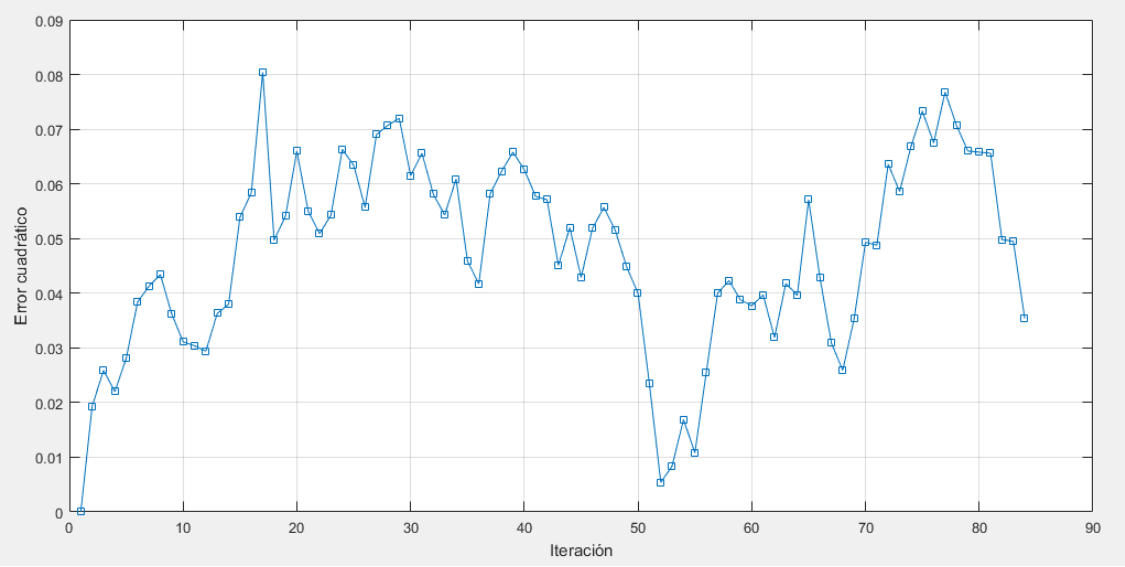


Figura A.14: Gráfica del error de pose en *MappingTB_v6_1*(0.5,0)

```
1 MappingTB_v6_1(0.8,0)
```

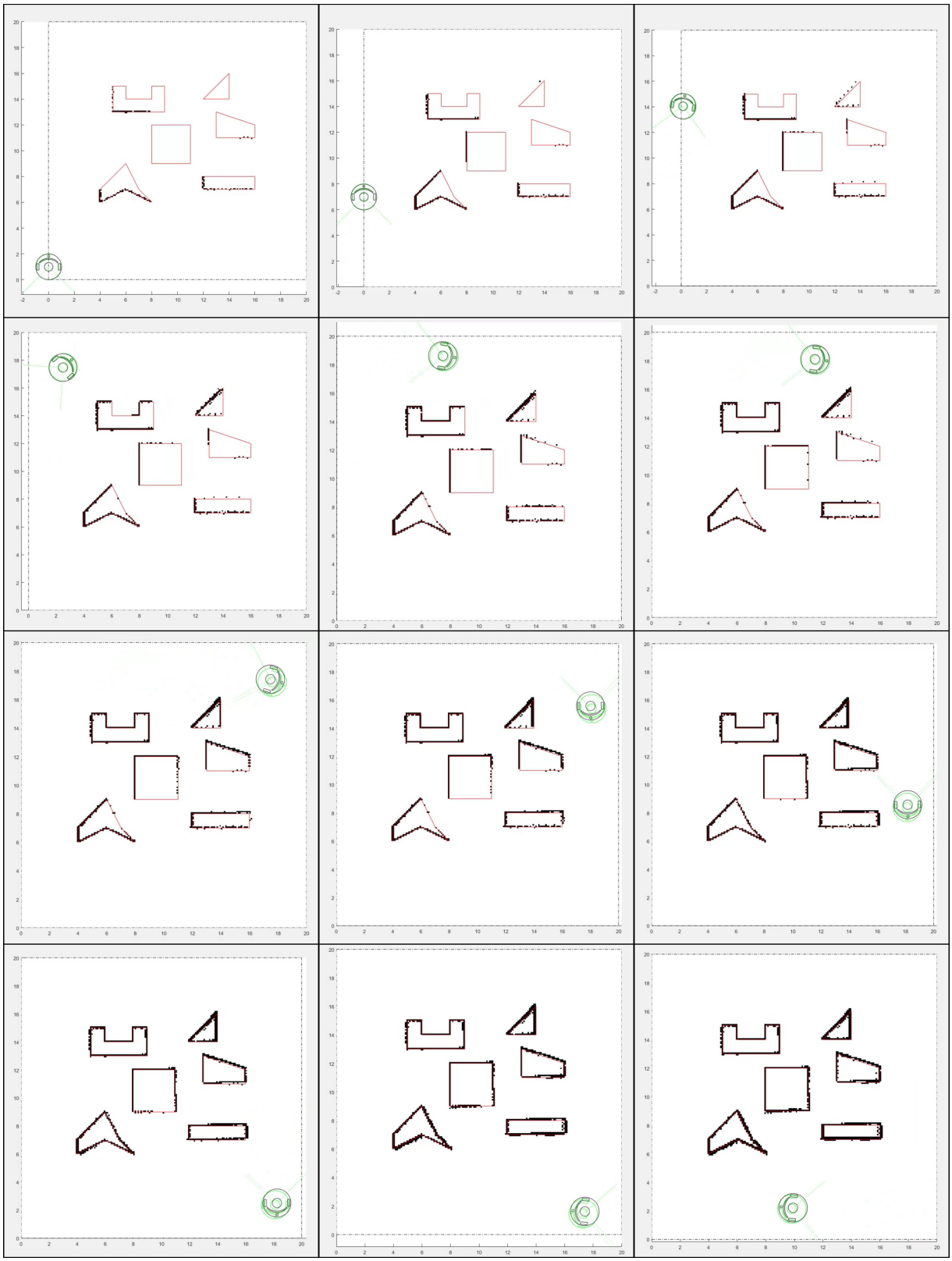


Figura A.15: Resultado de *MappingTB_v6_1* con error de odometría 0,8. Obstáculos en rojo, mapa en negro, pose real en verde y pose calculada en negro.

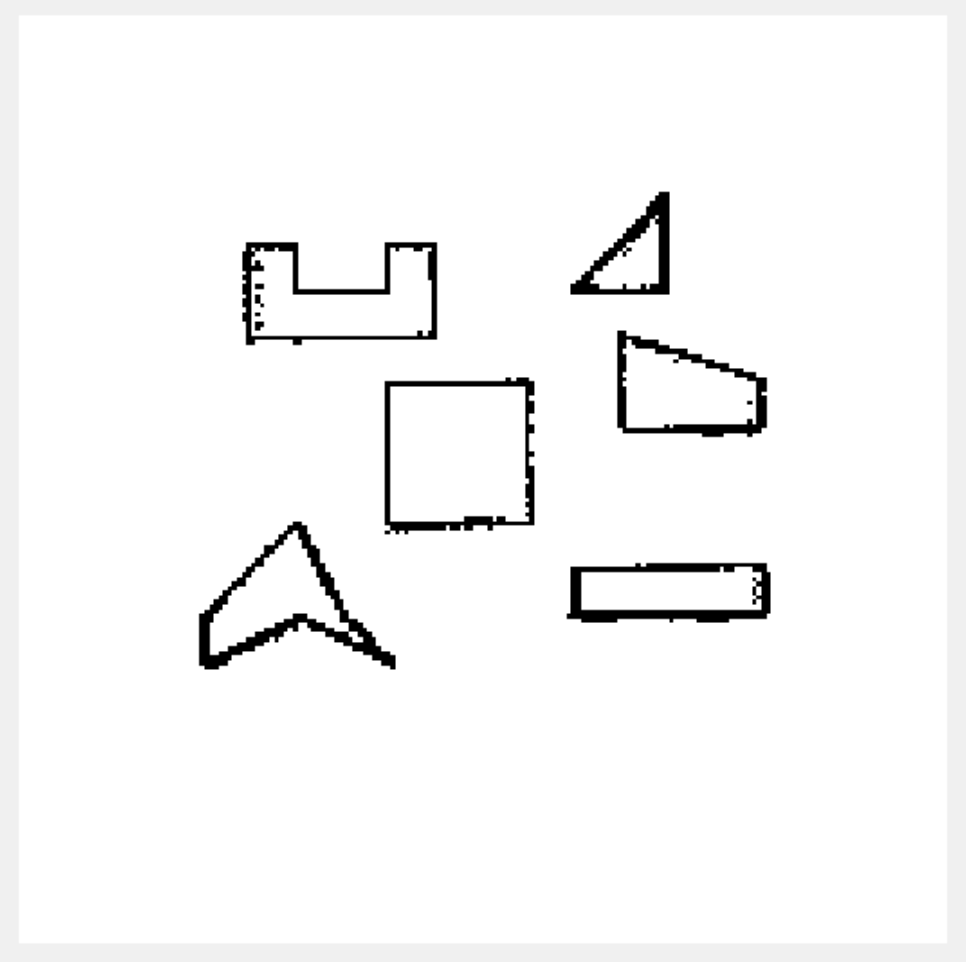


Figura A.16: Mapa resultante de *MappingTB_v6_1* con error de odometría 0,8

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0	1	0	0.0171
3	0	2	0	0.0306
4	0	3	0	0.018
5	0	4	0	0.0068
6	0	5	0	0.0207
7	0	6	0	0.0318
8	0	7	0	0.024
9	0	8	0	0.009
10	0	9	0	0.0294
11	0	10	0	0.042
12	0	11	0	0.042
13	0	12	0	0.0349
14	0	13	0	0.0414
15	0.179	14.04	-0.17	0.0473
16	0.532	15.04	-0.34	0.0296

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
17	1.049	15.96	-0.51	0.0194
18	1.714	16.79	-0.68	0.0259
19	2.509	17.49	-0.85	0.0465
20	3.411	18.04	-1.02	0.0728
21	4.394	18.43	-1.19	0.0776
22	5.428	18.66	-1.36	0.1031
23	6.486	18.7	-1.53	0.0971
24	7.535	18.56	-1.7	0.1032
25	8.527	18.43	-1.7	0.1058
26	9.519	18.3	-1.7	0.0949
27	10.51	18.18	-1.7	0.0854
28	11.5	18.05	-1.7	0.0935
29	12.49	17.92	-1.7	0.1216
30	13.48	17.79	-1.7	0.155
31	14.48	17.66	-1.7	0.1672
32	15.47	17.53	-1.7	0.1946
33	16.46	17.4	-1.7	0.2161
34	17.45	17.27	-1.7	0.2257
35	17.59	17.23	-1.88	0.237
36	17.73	17.16	-2.06	0.2275
37	17.84	17.06	-2.24	0.2329
38	17.94	16.95	-2.42	0.234
39	18.02	16.82	-2.6	0.2414
40	18.07	16.68	-2.78	0.2397
41	18.1	16.54	-2.96	0.2418
42	18.1	16.39	-3.14	0.2394
43	18.1	15.39	-3.14	0.2157
44	18.11	14.39	-3.14	0.2307
45	18.11	13.39	-3.14	0.1952
46	18.11	12.39	-3.14	0.2169
47	18.11	11.39	-3.14	0.2141
48	18.11	10.39	-3.14	0.1943
49	18.11	9.385	-3.14	0.1912
50	18.11	8.385	-3.14	0.2015
51	18.12	7.385	-3.14	0.2141
52	18.12	6.385	-3.14	0.192
53	18.12	5.385	-3.14	0.1879
54	18.12	4.385	-3.14	0.1712
55	18.12	3.385	-3.14	0.1766
56	18.12	2.385	-3.14	0.171
57	18.13	1.385	-3.14	0.1751
58	18.13	1.385	-2.48	0.1789
59	18.13	1.385	-1.82	0.1862
60	18.13	1.385	-1.16	0.1967
61	18.13	1.385	-0.5	0.1957

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
62	18.13	1.385	0.16	0.2024
63	18.13	1.385	0.82	0.2012
64	18.13	1.385	1.48	0.1974
65	17.3	1.46	1.48	0.2008
66	16.47	1.536	1.48	0.1916
67	15.64	1.611	1.48	0.1826
68	14.81	1.687	1.48	0.1528
69	13.98	1.763	1.48	0.161
70	13.15	1.838	1.48	0.1786
71	12.32	1.914	1.48	0.1668
72	11.49	1.989	1.48	0.1538
73	10.66	2.065	1.48	0.118
74	9.827	2.14	1.48	0.1159
75	8.997	2.216	1.48	0.0893
76	8.167	2.291	1.48	0.1016
77	7.337	2.367	1.48	0.1052
78	6.507	2.443	1.48	0.1141
79	5.677	2.518	1.48	0.1045
80	4.847	2.594	1.48	0.0758
81	4.017	2.669	1.48	0.0722
82	3.188	2.745	1.48	0.0822
83	2.358	2.82	1.48	0.1013
84	1.528	2.896	1.48	0.0949
Error cuadrático medio				0.1332

Tabla A.3: Error de pose en *MappingTB_v6_1*(0.8,0)

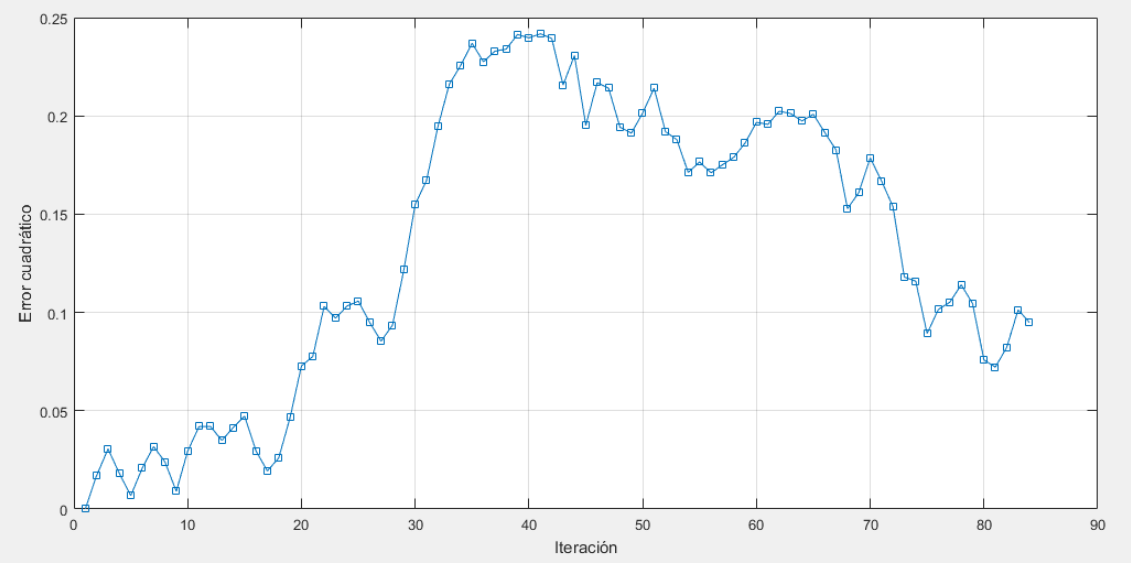


Figura A.17: Gráfica del error de pose en *MappingTB_v6_1*(0.8,0)

```
1 MappingTB_v6_1(1.1,0)
```

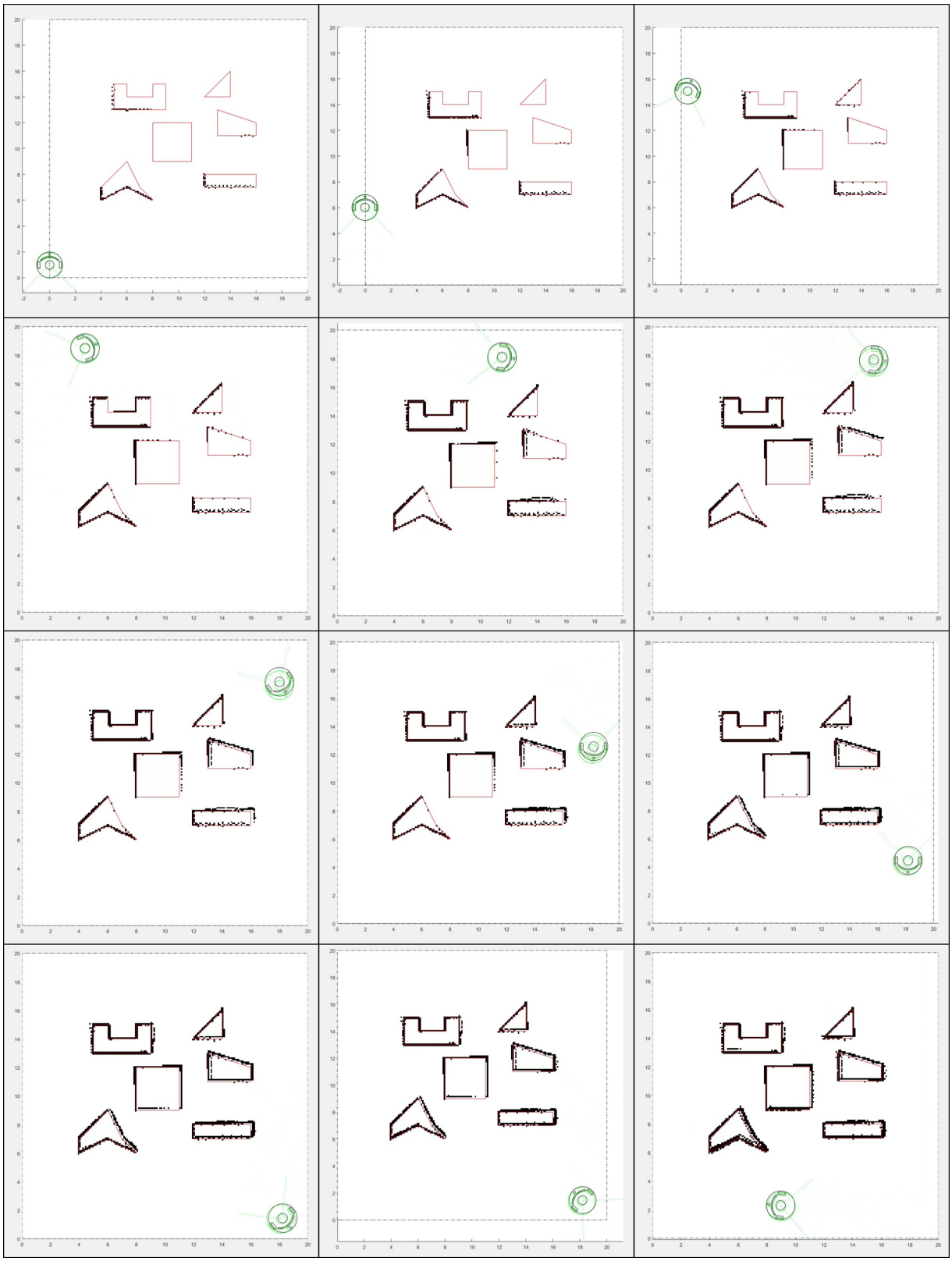


Figura A.18: Resultado de *MappingTB_v6_1* con error de odometría 1,1. Obstáculos en rojo, mapa en negro, pose real en verde y pose calculada en negro.

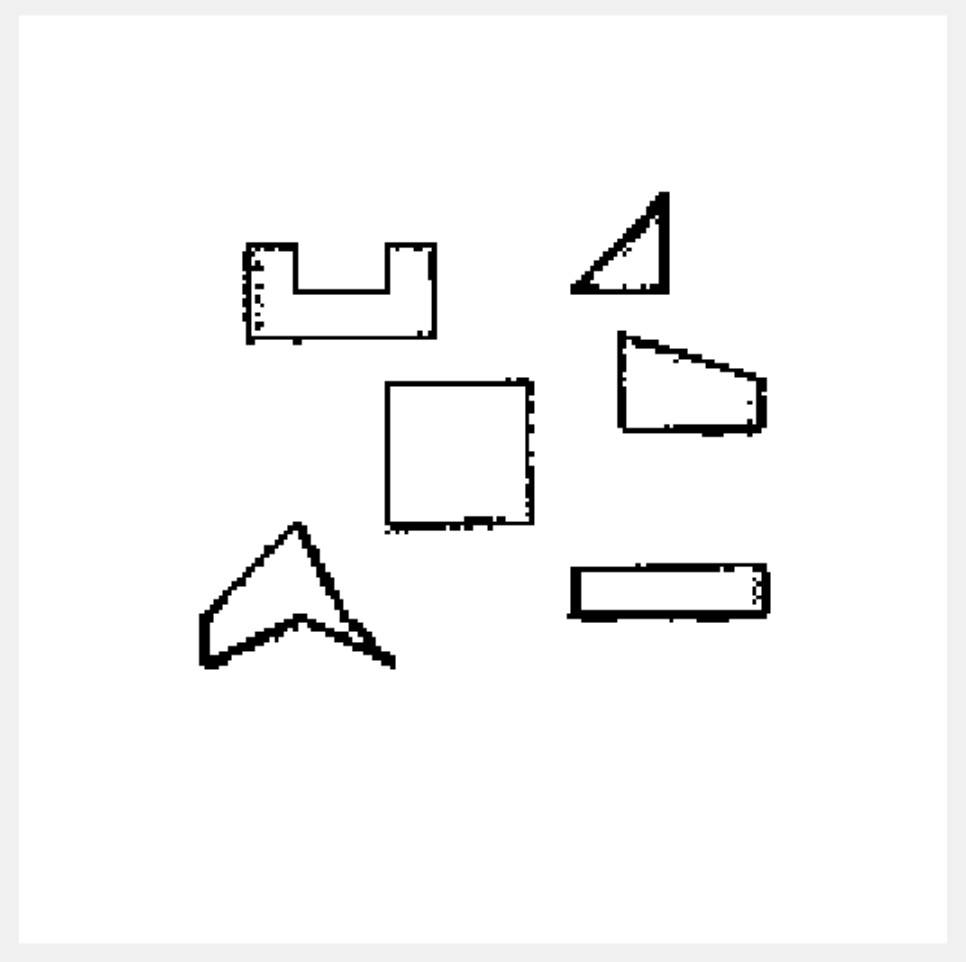


Figura A.19: Mapa resultante de *MappingTB_v6_1* con error de odometría 1,1

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0	1	0	0.0325
3	0	2	0	0.03
4	0	3	0	0.0501
5	0	4	0	0.0594
6	0	5	0	0.057
7	0	6	0	0.0495
8	0	7	0	0.0845
9	0	8	0	0.0572
10	0	9	0	0.0731
11	0	10	0	0.0697
12	0	11	0	0.0612
13	0	12	0	0.0857
14	0	13	0	0.0721
15	0.179	14.04	-0.17	0.0468
16	0.532	15.04	-0.34	0.0381

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
17	1.049	15.96	-0.51	0.0584
18	1.714	16.79	-0.68	0.0366
19	2.509	17.49	-0.85	0.0353
20	3.411	18.04	-1.02	0.0242
21	4.394	18.43	-1.19	0.0721
22	5.428	18.66	-1.36	0.0723
23	6.486	18.7	-1.53	0.0516
24	7.535	18.56	-1.7	0.0209
25	8.527	18.43	-1.7	0.0805
26	9.519	18.3	-1.7	0.1001
27	10.51	18.18	-1.7	0.0981
28	11.5	18.05	-1.7	0.0871
29	12.49	17.92	-1.7	0.0984
30	13.48	17.79	-1.7	0.1337
31	14.48	17.66	-1.7	0.1549
32	15.47	17.53	-1.7	0.1805
33	16.46	17.4	-1.7	0.2031
34	17.45	17.27	-1.7	0.2283
35	17.59	17.23	-1.88	0.2401
36	17.73	17.16	-2.06	0.2469
37	17.84	17.06	-2.24	0.2641
38	17.94	16.95	-2.42	0.2491
39	18.02	16.82	-2.6	0.2459
40	18.07	16.68	-2.78	0.2492
41	18.1	16.54	-2.96	0.2481
42	18.1	16.39	-3.14	0.2094
43	18.1	15.39	-3.14	0.2272
44	18.11	14.39	-3.14	0.2064
45	18.11	13.39	-3.14	0.2224
46	18.11	12.39	-3.14	0.2121
47	18.11	11.39	-3.14	0.2102
48	18.11	10.39	-3.14	0.2043
49	18.11	9.385	-3.14	0.1928
50	18.11	8.385	-3.14	0.2067
51	18.12	7.385	-3.14	0.168
52	18.12	6.385	-3.14	0.1348
53	18.12	5.385	-3.14	0.0947
54	18.12	4.385	-3.14	0.1177
55	18.12	3.385	-3.14	0.1388
56	18.12	2.385	-3.14	0.1465
57	18.13	1.385	-3.14	0.1386
58	18.13	1.385	-2.48	0.1433
59	18.13	1.385	-1.82	0.1294
60	18.13	1.385	-1.16	0.121
61	18.13	1.385	-0.5	0.0969

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
62	18.13	1.385	0.16	0.1088
63	18.13	1.385	0.82	0.1173
64	18.13	1.385	1.48	0.1196
65	17.3	1.46	1.48	0.1355
66	16.47	1.536	1.48	0.1069
67	15.64	1.611	1.48	0.1169
68	14.81	1.687	1.48	0.096
69	13.98	1.763	1.48	0.0985
70	13.15	1.838	1.48	0.0795
71	12.32	1.914	1.48	0.0955
72	11.49	1.989	1.48	0.1427
73	10.66	2.065	1.48	0.1156
74	9.827	2.14	1.48	0.0812
75	8.997	2.216	1.48	0.103
76	8.167	2.291	1.48	0.1273
77	7.337	2.367	1.48	0.1638
78	6.507	2.443	1.48	0.1587
79	5.677	2.518	1.48	0.1606
80	4.847	2.594	1.48	0.1641
81	4.017	2.669	1.48	0.1869
82	3.188	2.745	1.48	0.1855
83	2.358	2.82	1.48	0.2006
84	1.528	2.896	1.48	0.2305
Error cuadrático medio				0.1281

Tabla A.4: Error de pose en *MappingTB_v6_1*(1.1,0)

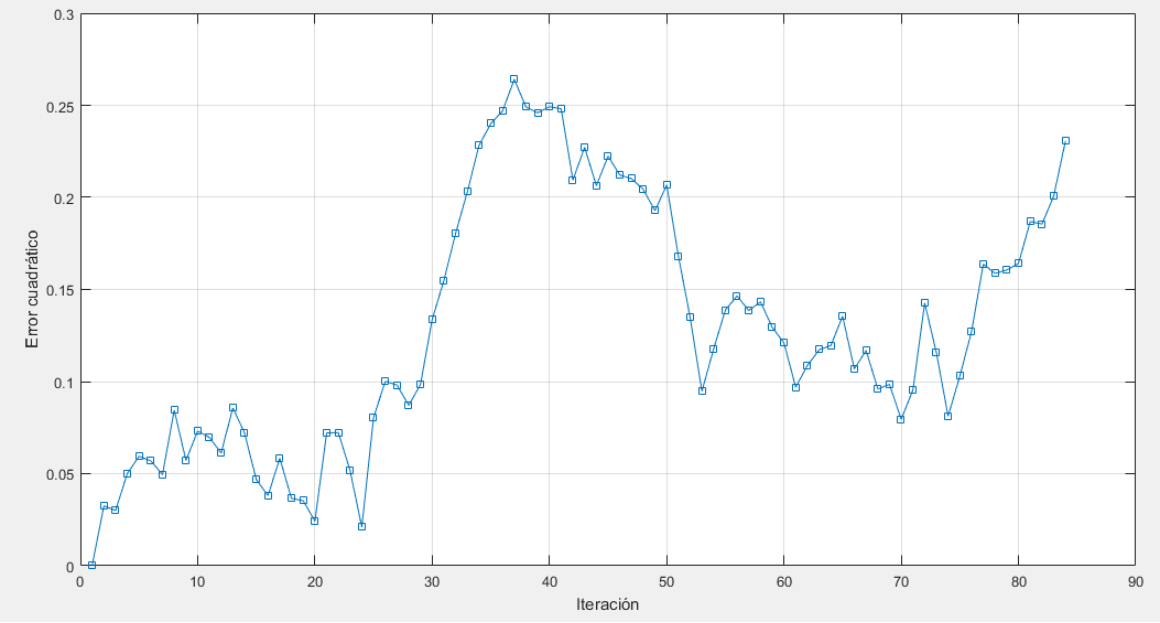


Figura A.20: Gráfica del error de pose en *MappingTB_v6_1*(1.1,0)

A.9. MappingTB_v6_2

```
1 MappingTB_v6_2(0.16,0)
```

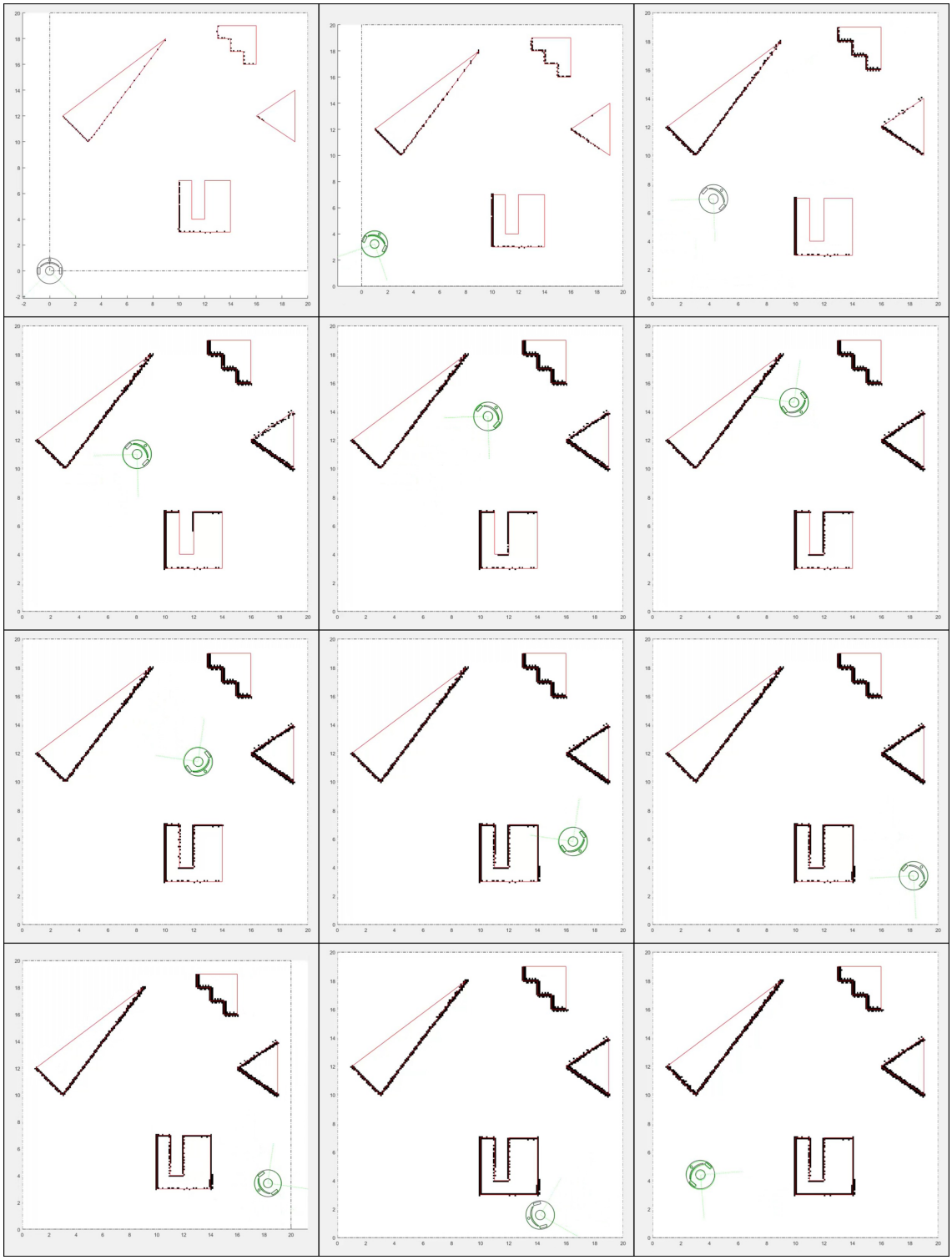


Figura A.21: Resultado de *MappingTB_v6_2* con error de odometría 0,16. Obstáculos en rojo, mapa en negro, pose real en verde y pose calculada en negro.

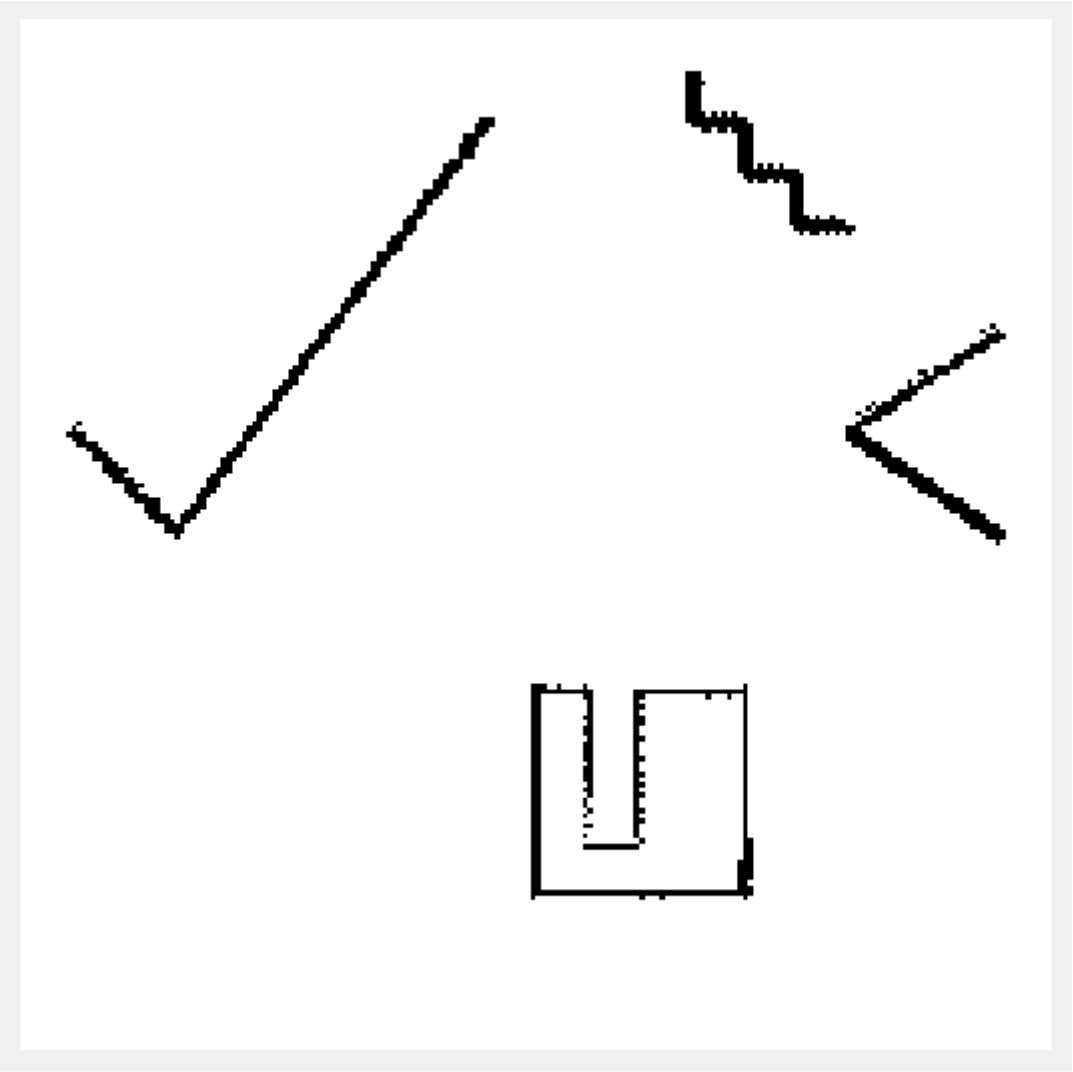


Figura A.22: Mapa resultante de *MappingTB_v6_2* con error de odometría 0,16

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0.168	1.112	-0.15	0.0213
3	0.501	2.187	-0.3	0.0213
4	0.99	3.2	-0.45	0.044
5	1.625	4.129	-0.6	0.0366
6	2.392	4.952	-0.75	0.0478
7	3.017	5.622	-0.75	0.048
8	3.642	6.293	-0.75	0.0135
9	4.266	6.964	-0.75	0.0071
10	4.891	7.635	-0.75	0.021
11	5.516	8.305	-0.75	0.0259
12	6.141	8.976	-0.75	0.0204
13	6.766	9.647	-0.75	0.0194
14	7.391	10.32	-0.75	0.0123

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
15	8.016	10.99	-0.75	0.0435
16	8.64	11.66	-0.75	0.0504
17	9.265	12.33	-0.75	0.0268
18	9.89	13	-0.75	0.0376
19	10.52	13.67	-0.75	0.0403
20	11.14	14.34	-0.75	0.06
21	10.88	14.21	-1.1	0.0481
22	10.59	14.17	-1.45	0.0514
23	10.31	14.24	-1.8	0.035
24	10.06	14.4	-2.15	0.0272
25	9.888	14.63	-2.5	0.0311
26	10.49	13.83	-2.5	0.0406
27	11.08	13.03	-2.5	0.0319
28	11.68	12.23	-2.5	0.0385
29	12.28	11.43	-2.5	0.0547
30	12.88	10.63	-2.5	0.064
31	13.48	9.827	-2.5	0.0714
32	14.08	9.026	-2.5	0.0576
33	14.68	8.225	-2.5	0.0615
34	15.27	7.424	-2.5	0.0624
35	15.87	6.622	-2.5	0.0625
36	16.47	5.821	-2.5	0.0421
37	17.07	5.02	-2.5	0.0243
38	17.67	4.219	-2.5	0.0214
39	18.27	3.418	-2.5	0.0259
40	18.27	3.418	-1.91	0.0215
41	18.27	3.418	-1.32	0.0196
42	18.27	3.418	-0.73	0.0173
43	18.27	3.418	-0.14	0.017
44	18.27	3.418	0.45	0.023
45	18.27	3.418	1.04	0.0161
46	18.27	3.418	1.63	0.0371
47	18.27	3.418	2.22	0.0232
48	17.51	2.931	2.145	0.0355
49	16.73	2.502	2.07	0.0201
50	15.91	2.134	1.995	0.0135
51	15.07	1.827	1.92	0.0266
52	14.21	1.585	1.845	0.0143
53	13.33	1.407	1.77	0.0194
54	12.44	1.296	1.695	0.0411
55	11.55	1.252	1.62	0.0441
56	10.65	1.275	1.545	0.04
57	9.758	1.366	1.47	0.0304
58	8.876	1.522	1.395	0.052
59	8.008	1.745	1.32	0.0697

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
60	7.16	2.031	1.245	0.07
61	6.335	2.381	1.17	0.0884
62	5.539	2.791	1.095	0.074
63	4.775	3.26	1.02	0.0617
64	4.049	3.785	0.945	0.0593
65	3.364	4.362	0.87	0.0652
66	2.725	4.99	0.795	0.0619
67	2.134	5.663	0.72	0.0652
Error cuadrático medio				0.03844

Tabla A.5: Error de pose en *MappingTB_v6_2*(0.16,0)

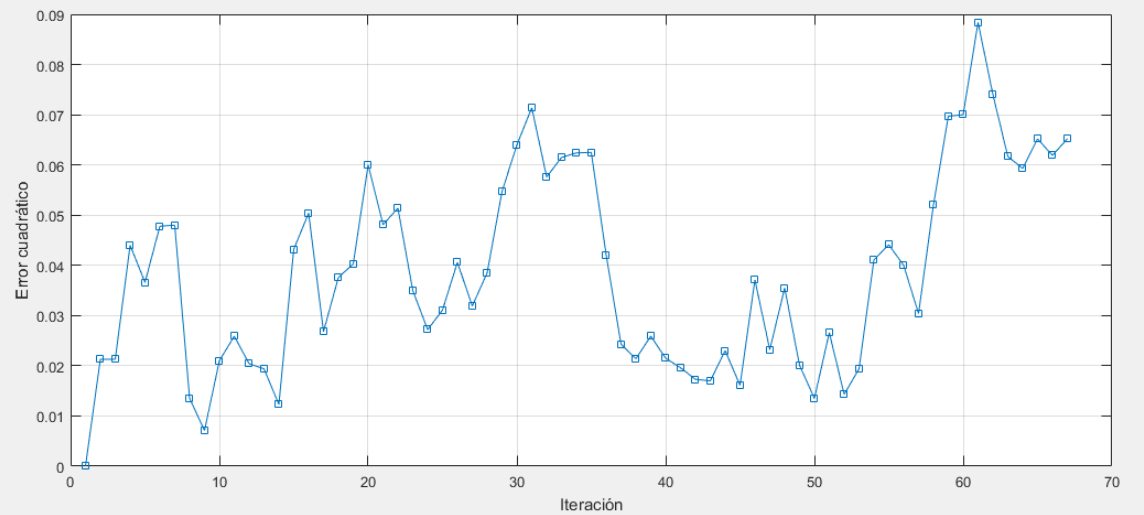


Figura A.23: Gráfica del error de pose en *MappingTB_v6_2*(0.16,0)

```
1 MappingTB_v6_2(0.3,0)
```

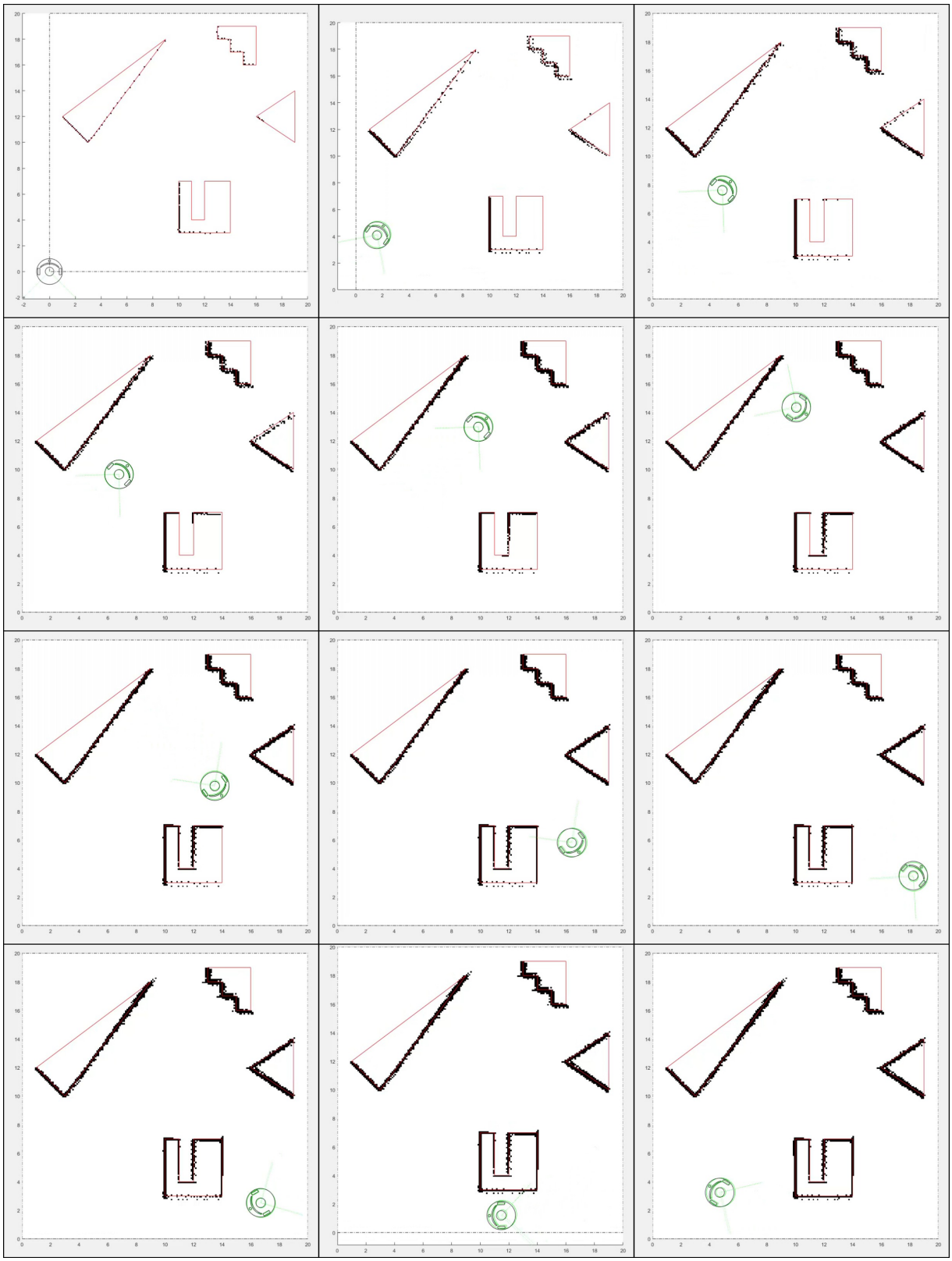


Figura A.24: Resultado de *MappingTB_v6_2* con error de odometría 0,3. Obstáculos en rojo, mapa en negro, pose real en verde y pose calculada en negro.

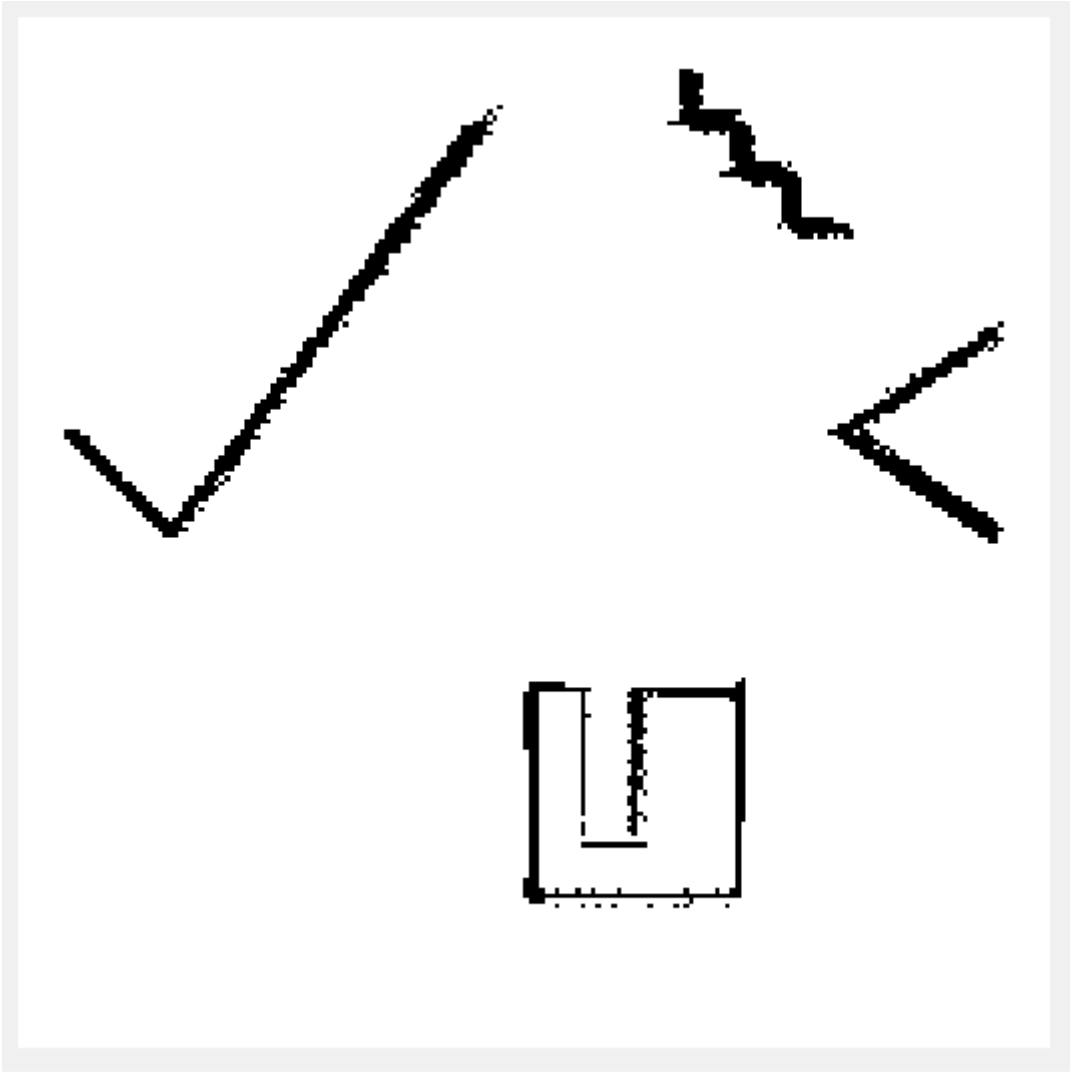


Figura A.25: Mapa resultante de *MappingTB_v6_2* con error de odometría 0,3

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0.168	1.112	-0.15	0.075
3	0.501	2.187	-0.3	0.0708
4	0.99	3.2	-0.45	0.0912
5	1.625	4.129	-0.6	0.086
6	2.392	4.952	-0.75	0.0615
7	3.017	5.622	-0.75	0.0851
8	3.642	6.293	-0.75	0.0799
9	4.266	6.964	-0.75	0.0459
10	4.891	7.635	-0.75	0.0436
11	5.516	8.305	-0.75	0.024
12	6.141	8.976	-0.75	0.0346
13	6.766	9.647	-0.75	0.0327
14	7.391	10.32	-0.75	0.0526

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
15	8.016	10.99	-0.75	0.0921
16	8.64	11.66	-0.75	0.1166
17	9.265	12.33	-0.75	0.0596
18	9.89	13	-0.75	0.0484
19	10.52	13.67	-0.75	0.0733
20	11.14	14.34	-0.75	0.058
21	10.88	14.21	-1.1	0.0634
22	10.59	14.17	-1.45	0.0636
23	10.31	14.24	-1.8	0.0724
24	10.06	14.4	-2.15	0.0792
25	9.888	14.63	-2.5	0.0505
26	10.49	13.83	-2.5	0.076
27	11.08	13.03	-2.5	0.0597
28	11.68	12.23	-2.5	0.0317
29	12.28	11.43	-2.5	0.0529
30	12.88	10.63	-2.5	0.0612
31	13.48	9.827	-2.5	0.0705
32	14.08	9.026	-2.5	0.0794
33	14.68	8.225	-2.5	0.0921
34	15.27	7.424	-2.5	0.0653
35	15.87	6.622	-2.5	0.058
36	16.47	5.821	-2.5	0.0829
37	17.07	5.02	-2.5	0.06
38	17.67	4.219	-2.5	0.0788
39	18.27	3.418	-2.5	0.097
40	18.27	3.418	-1.91	0.1028
41	18.27	3.418	-1.32	0.0947
42	18.27	3.418	-0.73	0.0808
43	18.27	3.418	-0.14	0.0725
44	18.27	3.418	0.45	0.066
45	18.27	3.418	1.04	0.0421
46	18.27	3.418	1.63	0.0529
47	18.27	3.418	2.22	0.0603
48	17.51	2.931	2.145	0.0828
49	16.73	2.502	2.07	0.0435
50	15.91	2.134	1.995	0.038
51	15.07	1.827	1.92	0.0631
52	14.21	1.585	1.845	0.0976
53	13.33	1.407	1.77	0.0515
54	12.44	1.296	1.695	0.0763
55	11.55	1.252	1.62	0.1026
56	10.65	1.275	1.545	0.1167
57	9.758	1.366	1.47	0.1308
58	8.876	1.522	1.395	0.1167
59	8.008	1.745	1.32	0.1318

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
60	7.16	2.031	1.245	0.1028
61	6.335	2.381	1.17	0.1151
62	5.539	2.791	1.095	0.0984
63	4.775	3.26	1.02	0.0882
64	4.049	3.785	0.945	0.0887
65	3.364	4.362	0.87	0.0751
66	2.725	4.99	0.795	0.0858
67	2.134	5.663	0.72	0.0749
Error cuadrático medio				0.07326

Tabla A.6: Error de pose en *MappingTB_v6_2*(0.3,0)

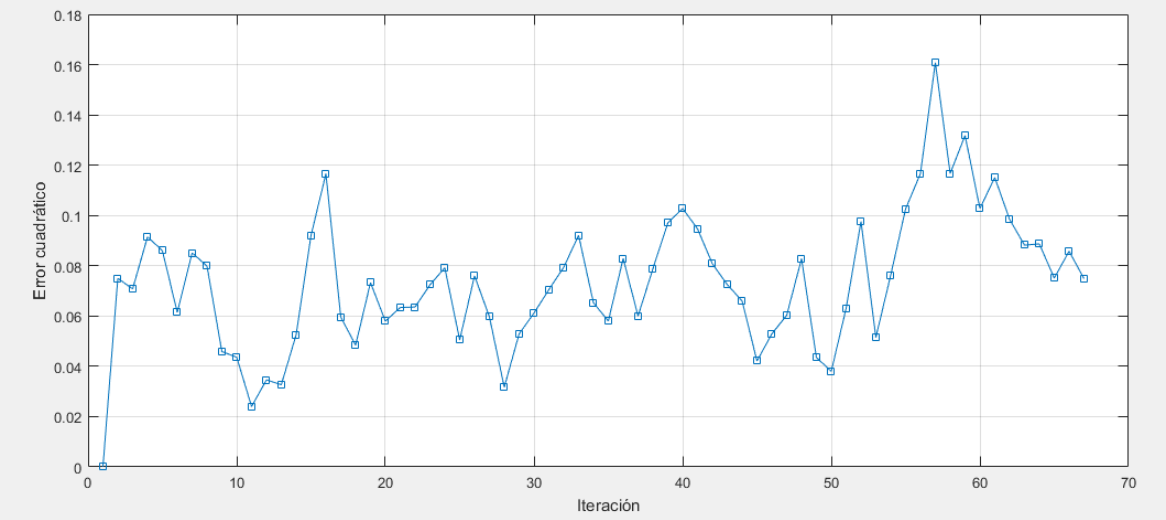


Figura A.26: Gráfica del error de pose en *MappingTB_v6_2*(0.3,0)

```
1 MappingTB_v6_2(0.3,0.1)
```

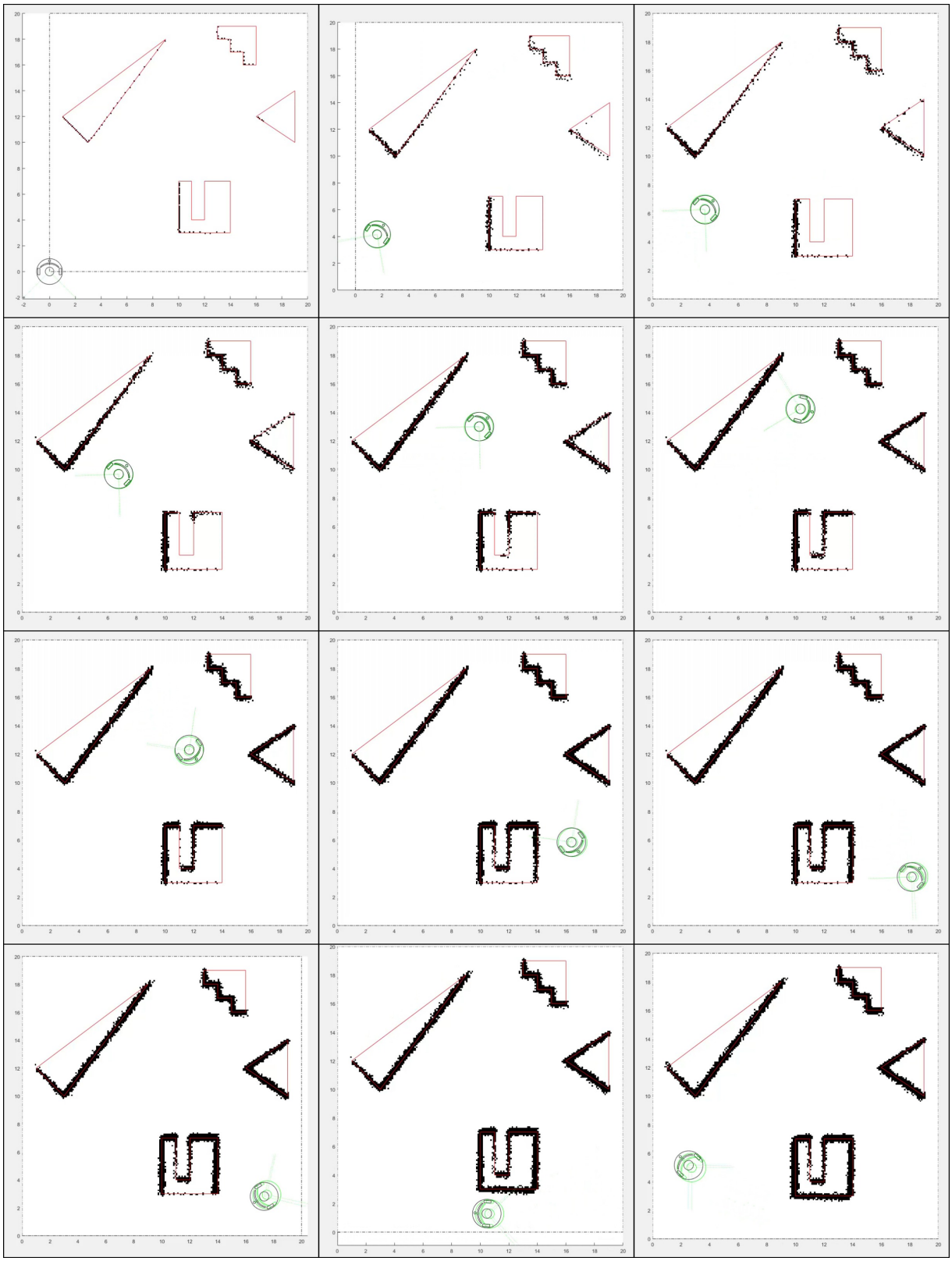


Figura A.27: Resultado de *MappingTB_v6_2* con error de odometría 0,3 y de distancia 0,1. Obstáculos en rojo, mapa en negro, pose real en verde y pose calculada en negro.

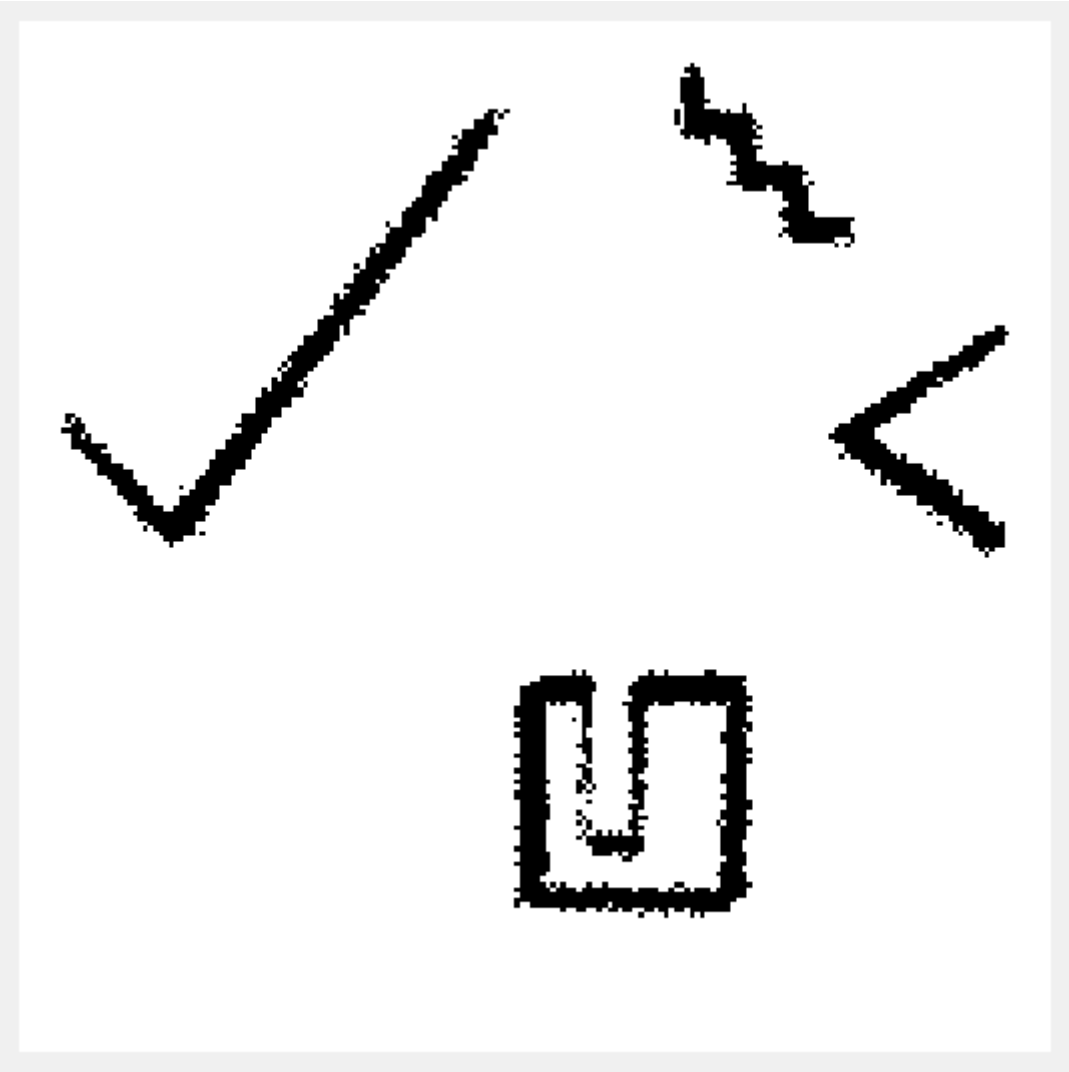


Figura A.28: Mapa resultante de *MappingTB_v6_2* con error de odometría 0,3 y de distancia 0,1

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0.168	1.112	-0.15	0.0357
3	0.501	2.187	-0.3	0.0413
4	0.99	3.2	-0.45	0.0599
5	1.625	4.129	-0.6	0.0404
6	2.392	4.952	-0.75	0.0258
7	3.017	5.622	-0.75	0.0308
8	3.642	6.293	-0.75	0.0527
9	4.266	6.964	-0.75	0.0434
10	4.891	7.635	-0.75	0.0614
11	5.516	8.305	-0.75	0.0292
12	6.141	8.976	-0.75	0.0407
13	6.766	9.647	-0.75	0.034
14	7.391	10.32	-0.75	0.0234

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
15	8.016	10.99	-0.75	0.0634
16	8.64	11.66	-0.75	0.0139
17	9.265	12.33	-0.75	0.0806
18	9.89	13	-0.75	0.0582
19	10.52	13.67	-0.75	0.0516
20	11.14	14.34	-0.75	0.0475
21	10.88	14.21	-1.1	0.052
22	10.59	14.17	-1.45	0.0532
23	10.31	14.24	-1.8	0.0635
24	10.06	14.4	-2.15	0.052
25	9.888	14.63	-2.5	0.0867
26	10.49	13.83	-2.5	0.0867
27	11.08	13.03	-2.5	0.0923
28	11.68	12.23	-2.5	0.0995
29	12.28	11.43	-2.5	0.1047
30	12.88	10.63	-2.5	0.0978
31	13.48	9.827	-2.5	0.0946
32	14.08	9.026	-2.5	0.1183
33	14.68	8.225	-2.5	0.1019
34	15.27	7.424	-2.5	0.0997
35	15.87	6.622	-2.5	0.0954
36	16.47	5.821	-2.5	0.0968
37	17.07	5.02	-2.5	0.1149
38	17.67	4.219	-2.5	0.1339
39	18.27	3.418	-2.5	0.1653
40	18.27	3.418	-1.91	0.1369
41	18.27	3.418	-1.32	0.1589
42	18.27	3.418	-0.73	0.1448
43	18.27	3.418	-0.14	0.1674
44	18.27	3.418	0.45	0.1691
45	18.27	3.418	1.04	0.1899
46	18.27	3.418	1.63	0.1545
47	18.27	3.418	2.22	0.183
48	17.51	2.931	2.145	0.2211
49	16.73	2.502	2.07	0.2399
50	15.91	2.134	1.995	0.2768
51	15.07	1.827	1.92	0.2689
52	14.21	1.585	1.845	0.2534
53	13.33	1.407	1.77	0.2104
54	12.44	1.296	1.695	0.1984
55	11.55	1.252	1.62	0.2096
56	10.65	1.275	1.545	0.1936
57	9.758	1.366	1.47	0.1838
58	8.876	1.522	1.395	0.2347
59	8.008	1.745	1.32	0.1884

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
60	7.16	2.031	1.245	0.2426
61	6.335	2.381	1.17	0.2125
62	5.539	2.791	1.095	0.2257
63	4.775	3.26	1.02	0.2204
64	4.049	3.785	0.945	0.2387
65	3.364	4.362	0.87	0.2452
66	2.725	4.99	0.795	0.2686
67	2.134	5.663	0.72	0.2785
Error cuadrático medio				0.1277

Tabla A.7: Error de pose en *MappingTB_v6_2*(0.3,0.1)

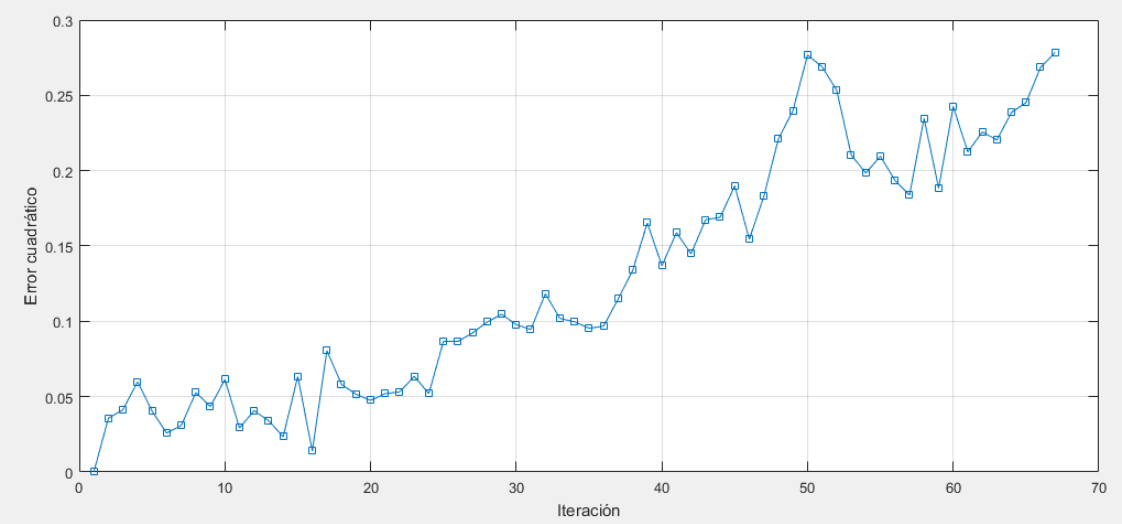


Figura A.29: Gráfica del error de pose en *MappingTB_v6_2*(0.3,0.1)

A.10. MappingTB_v6_3

```
1 MappingTB_v6_3(0,0)
```

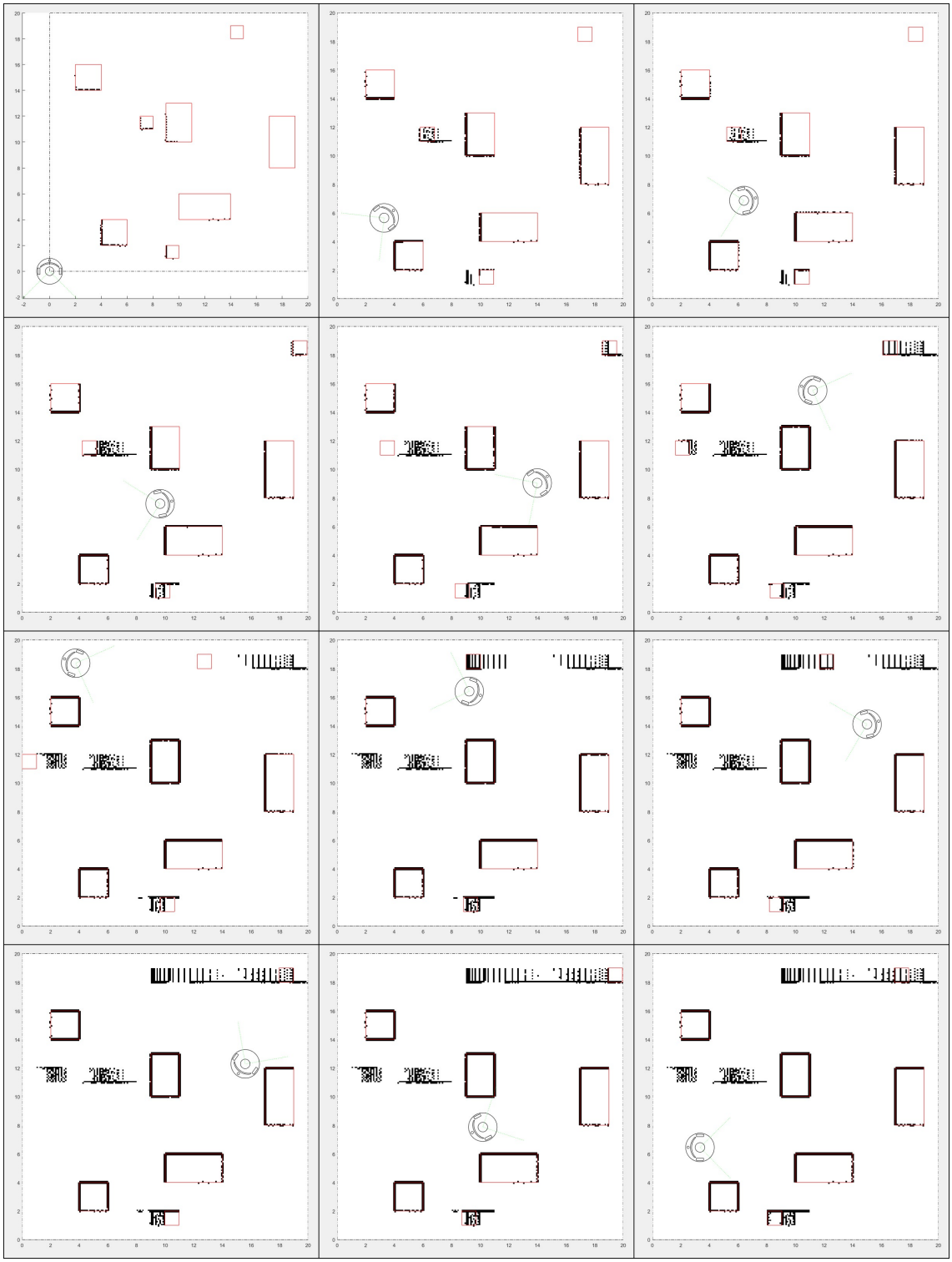


Figura A.30: Resultado de *MappingTB_v7_3* sin error. Obstáculos en rojo, mapa negro y pose en negro.

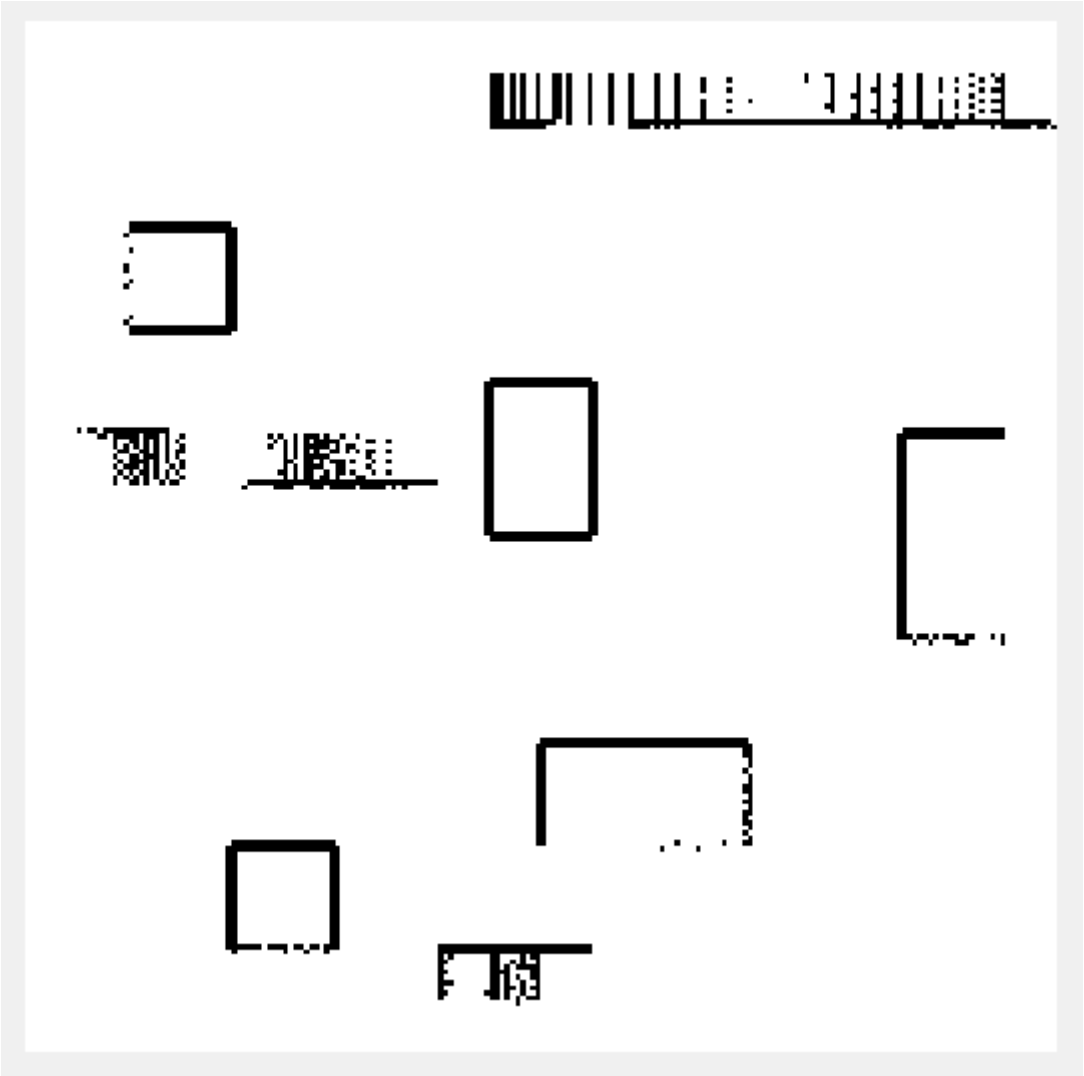


Figura A.31: Mapa resultante de *MappingTB_v6_3(0,0)*

A.11. MappingTB_v7_1

```
1 MappingTB_v7_1(0,0)
```

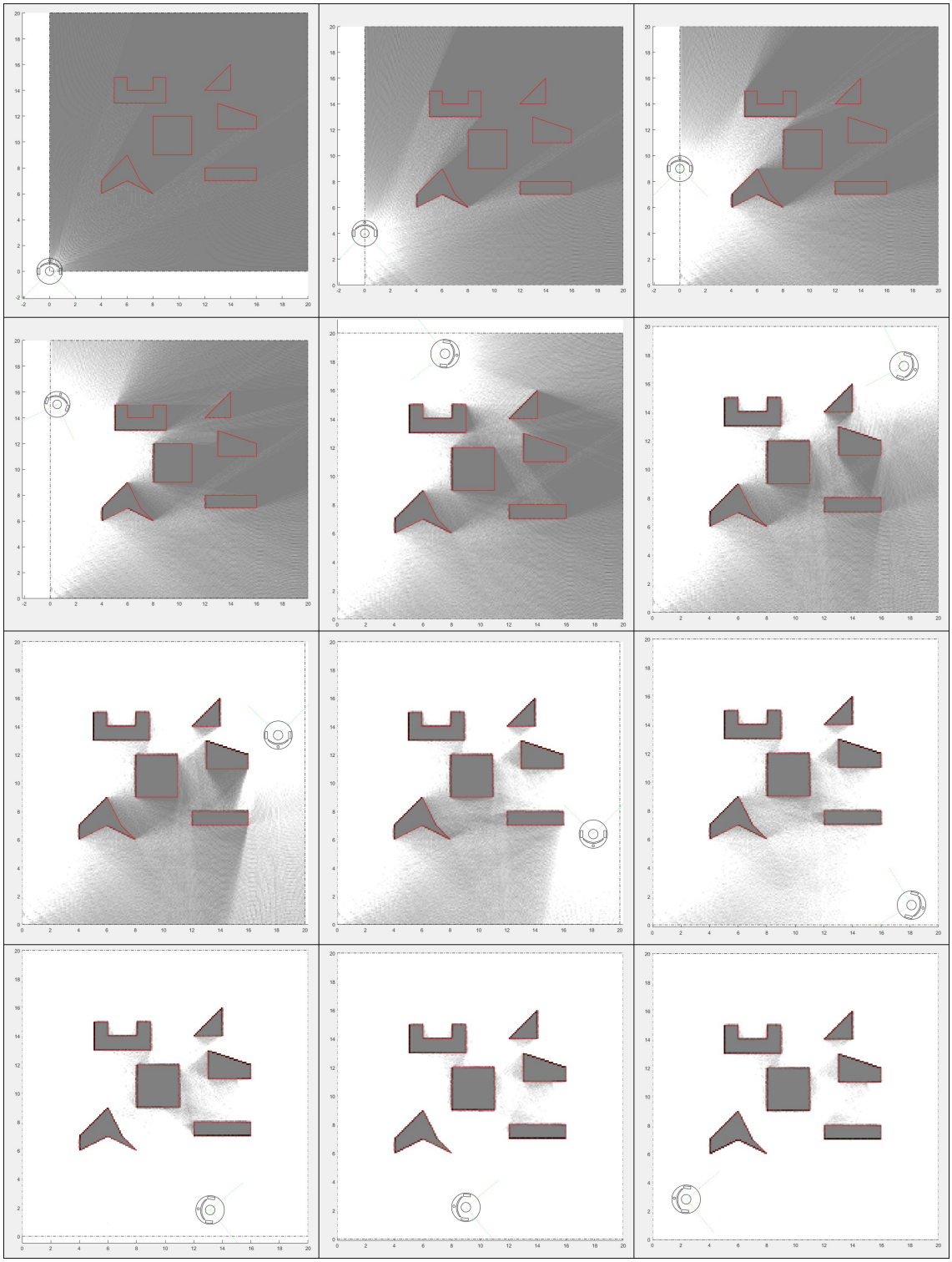


Figura A.32: Resultado de *MappingTB_v7_1* sin error. Obstáculos en rojo, mapa en escala de grises, pose real en verde y pose calculada en negro.

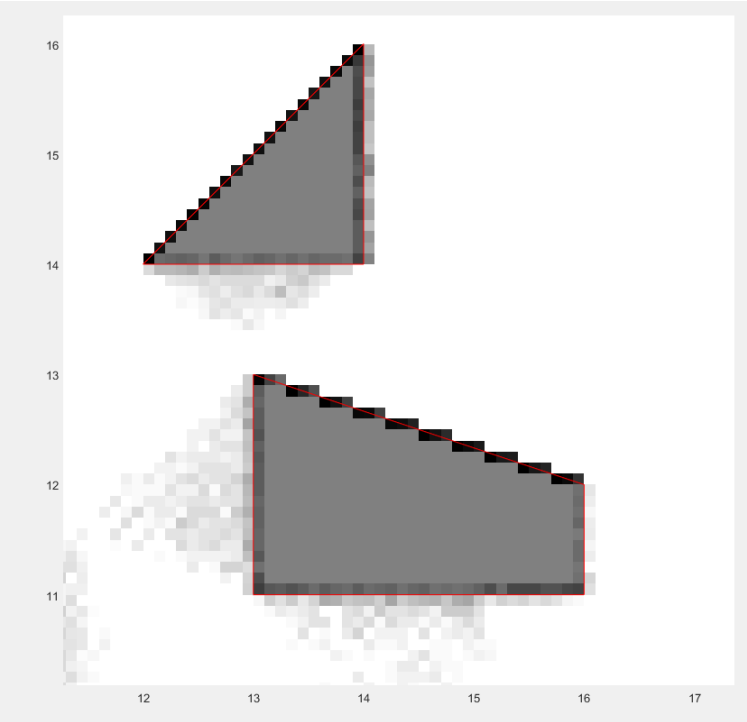


Figura A.33: Ampliación del resultado final de *MappingTB_v7_1* sin error

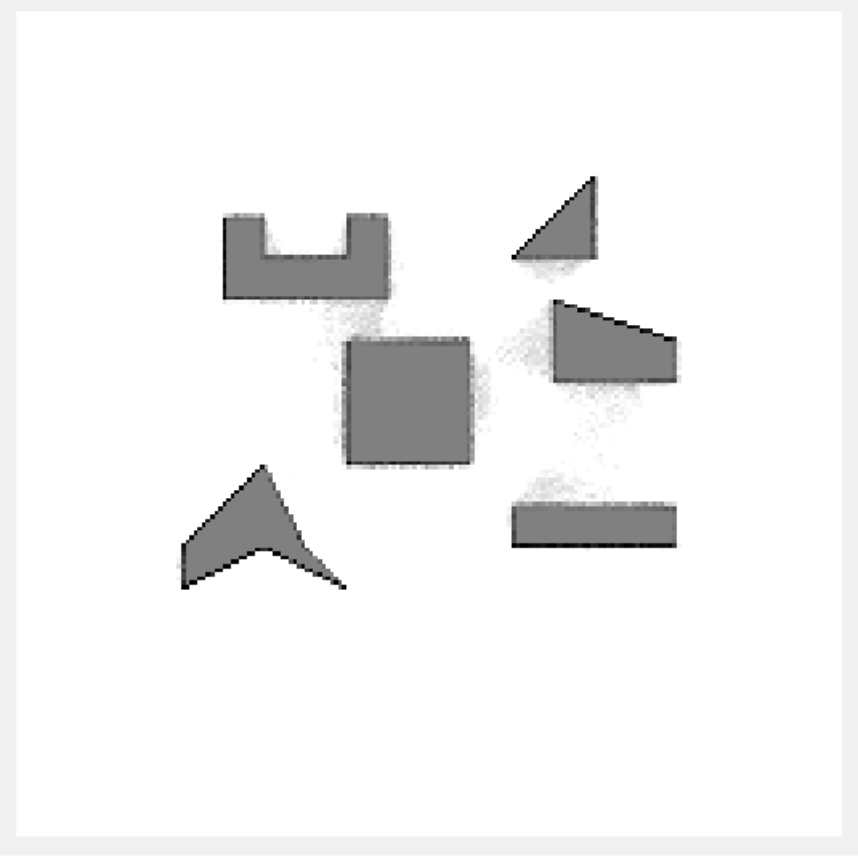


Figura A.34: Mapa resultante de *MappingTB_v7_1(0,0)*

```
1 MappingTB_v7_1(0,0.1)
```

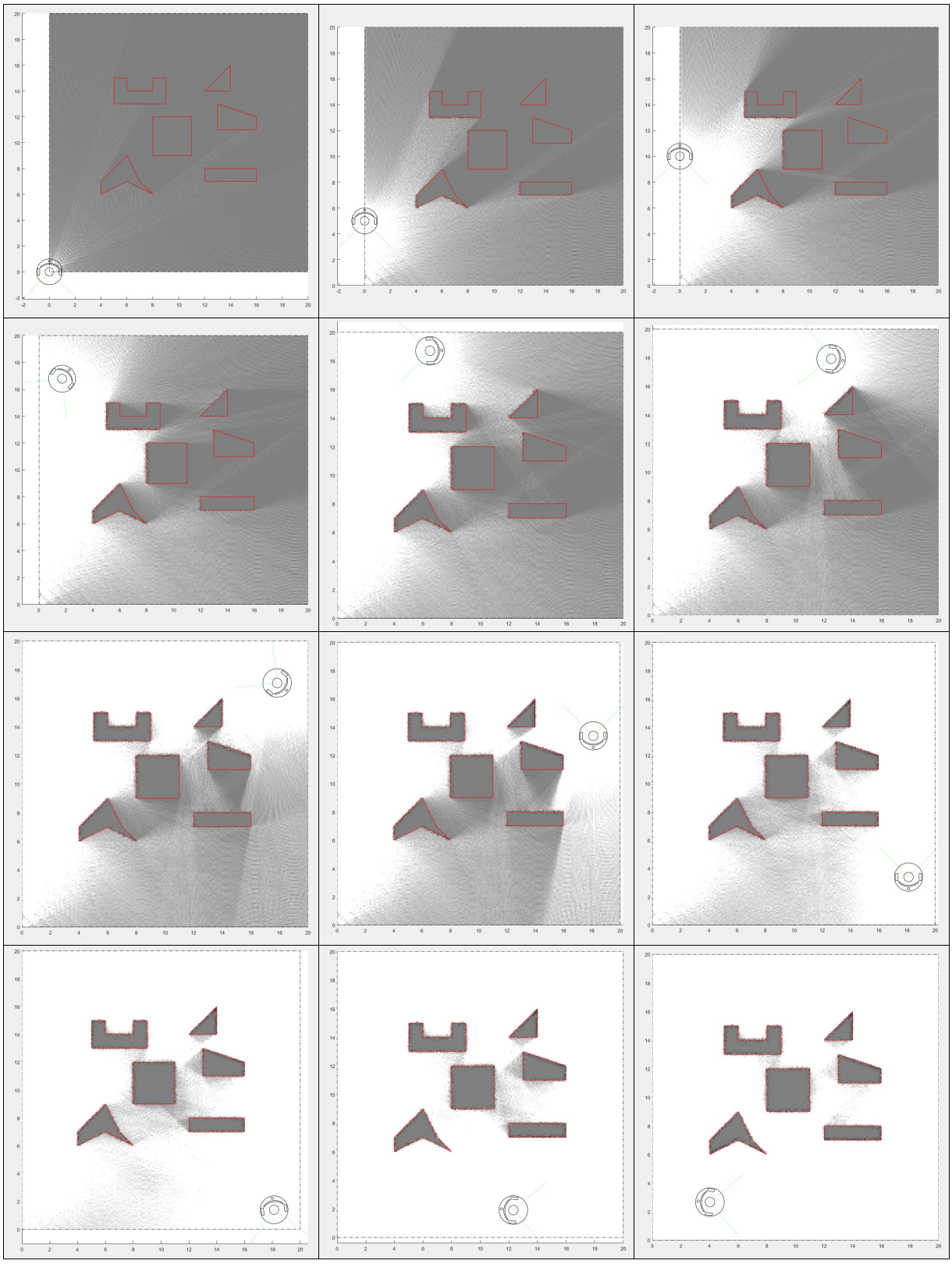


Figura A.35: Resultado de *MappingTB_v7_1* con error de distancia 0.1. Obstáculos en rojo, mapa en escala de grises y pose en negro.

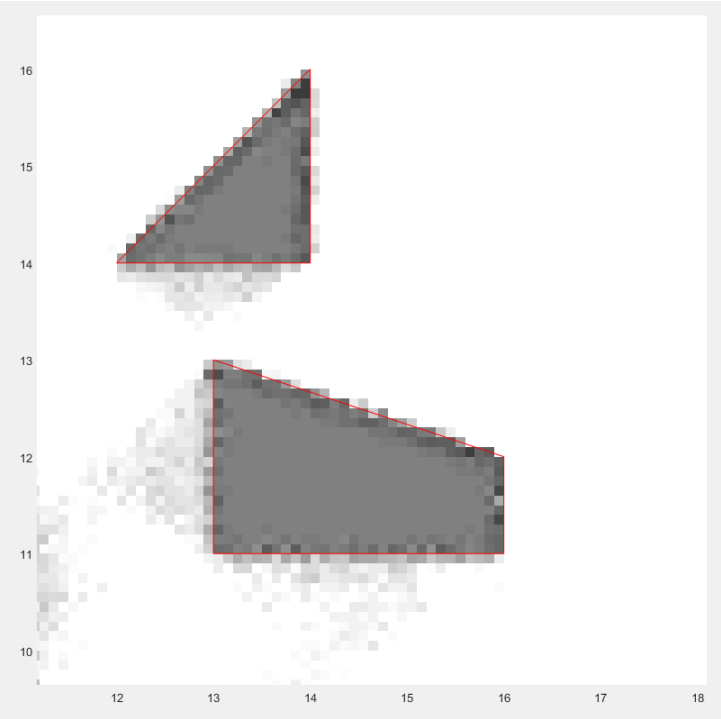


Figura A.36: Ampliación del resultado final de *MappingTB_v7_1* con error de distancia 0.1

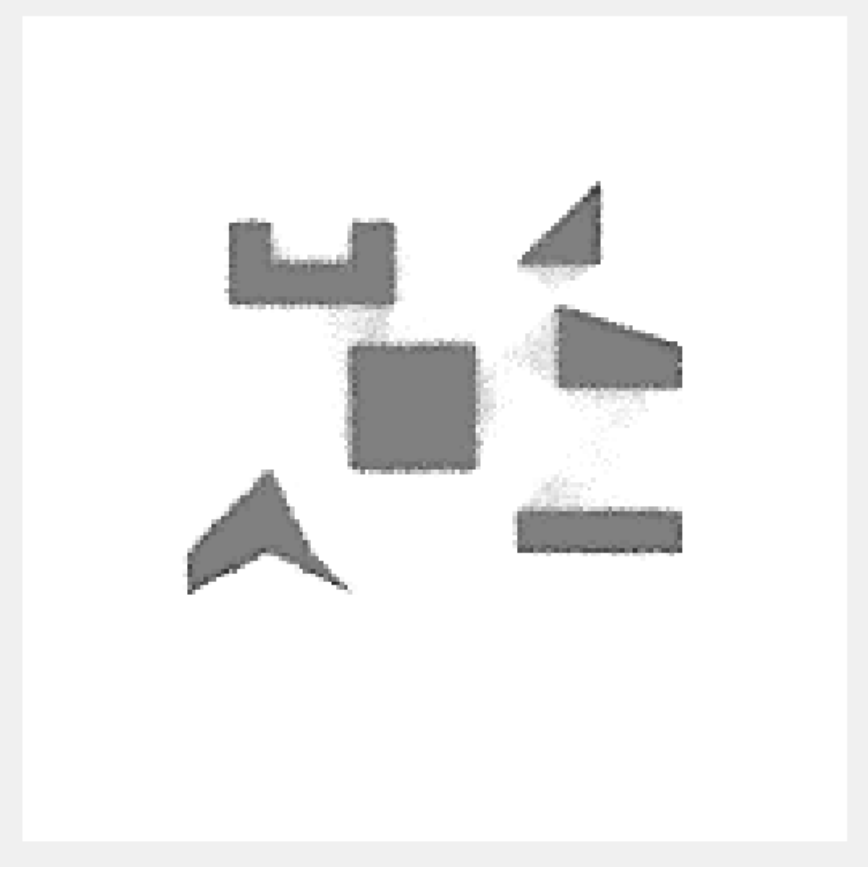


Figura A.37: Mapa resultante de *MappingTB_v7_1(0,0.1)*

A.12. MappingTB_v7_2

```
1 MappingTB_v7_2(0,0.1)
```

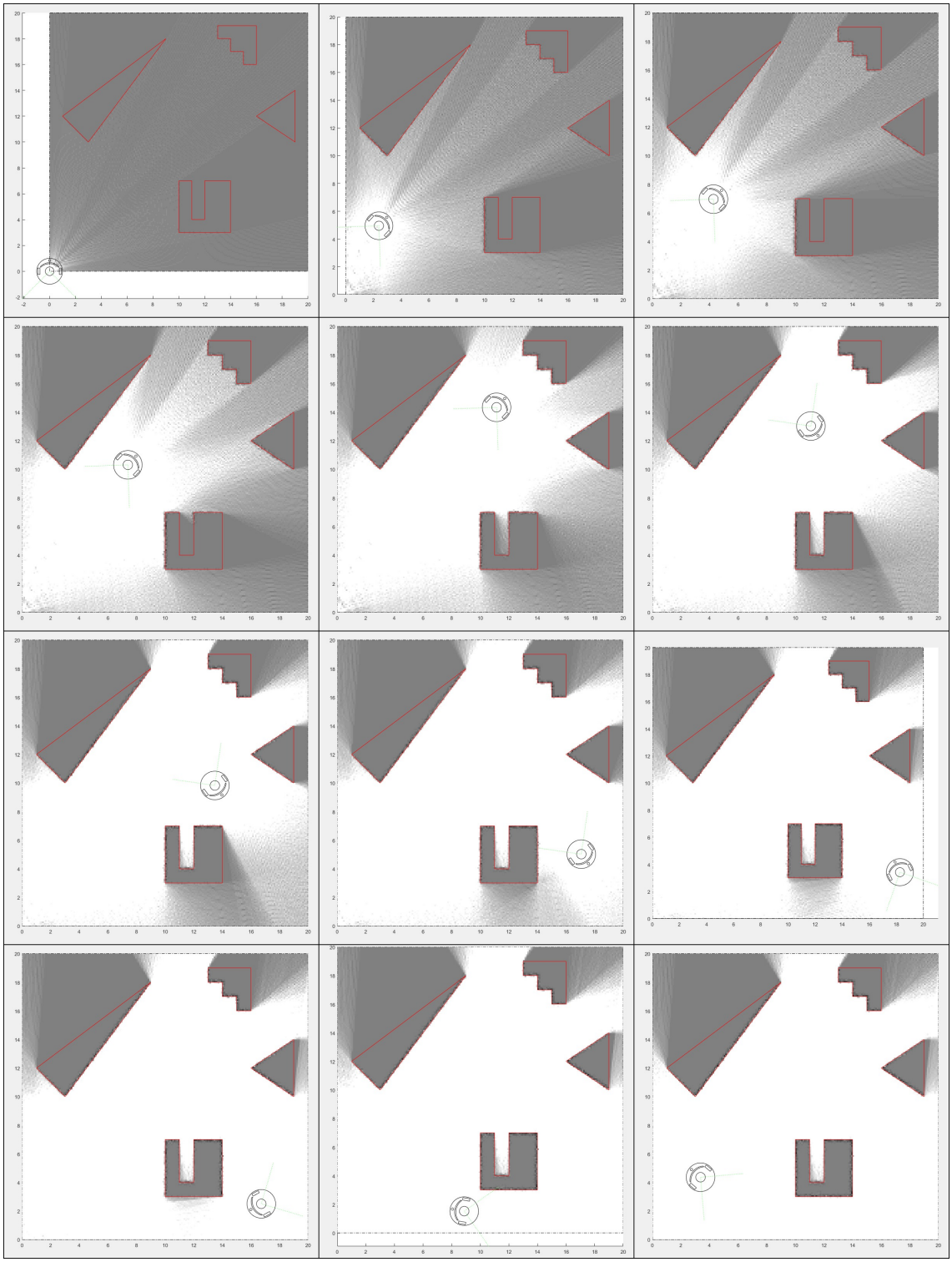


Figura A.38: Resultado de *MappingTB_v7_2* con error de distancia 0.1. Obstáculos en rojo, mapa en escala de grises y pose en negro.

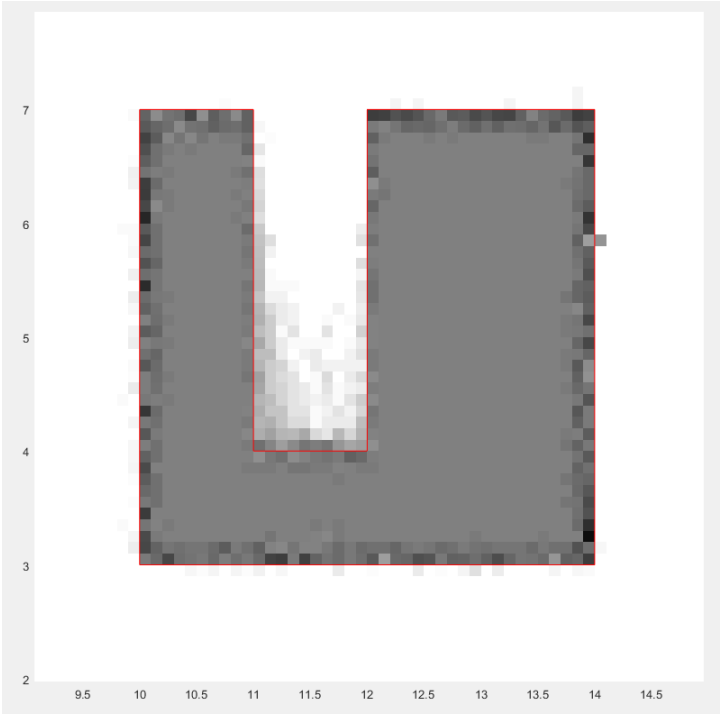


Figura A.39: Ampliación del resultado final de *MappingTB_v7_2* con error de distancia 0.1

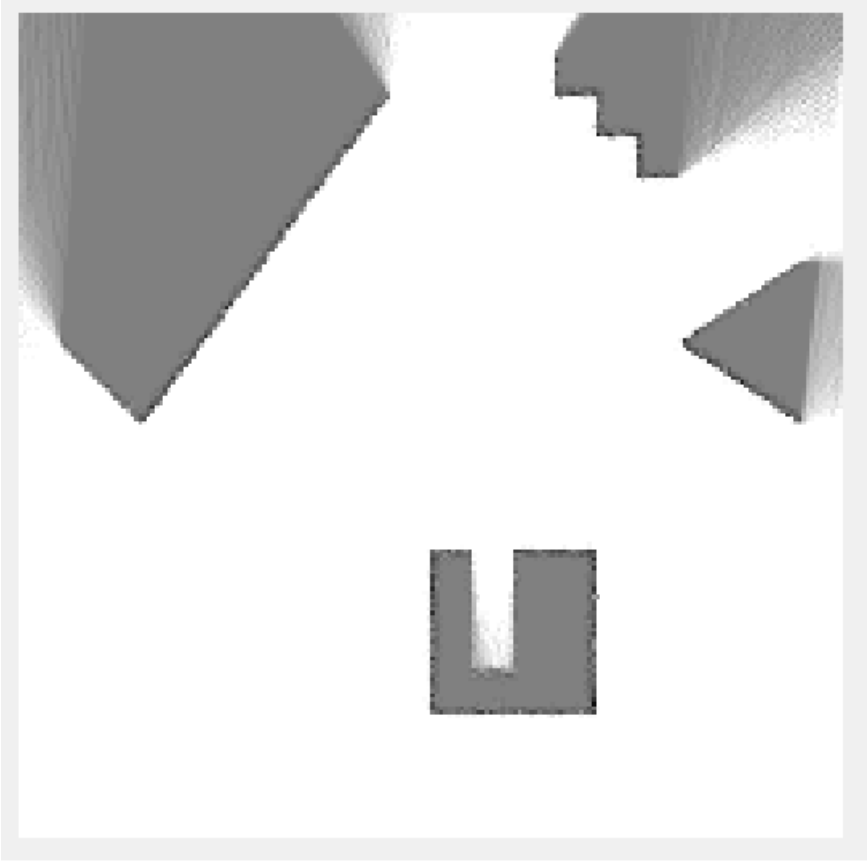


Figura A.40: Mapa resultante de *MappingTB_v7_2(0,0.1)*

```
1 MappingTB_v7_2(0,0.2)
```

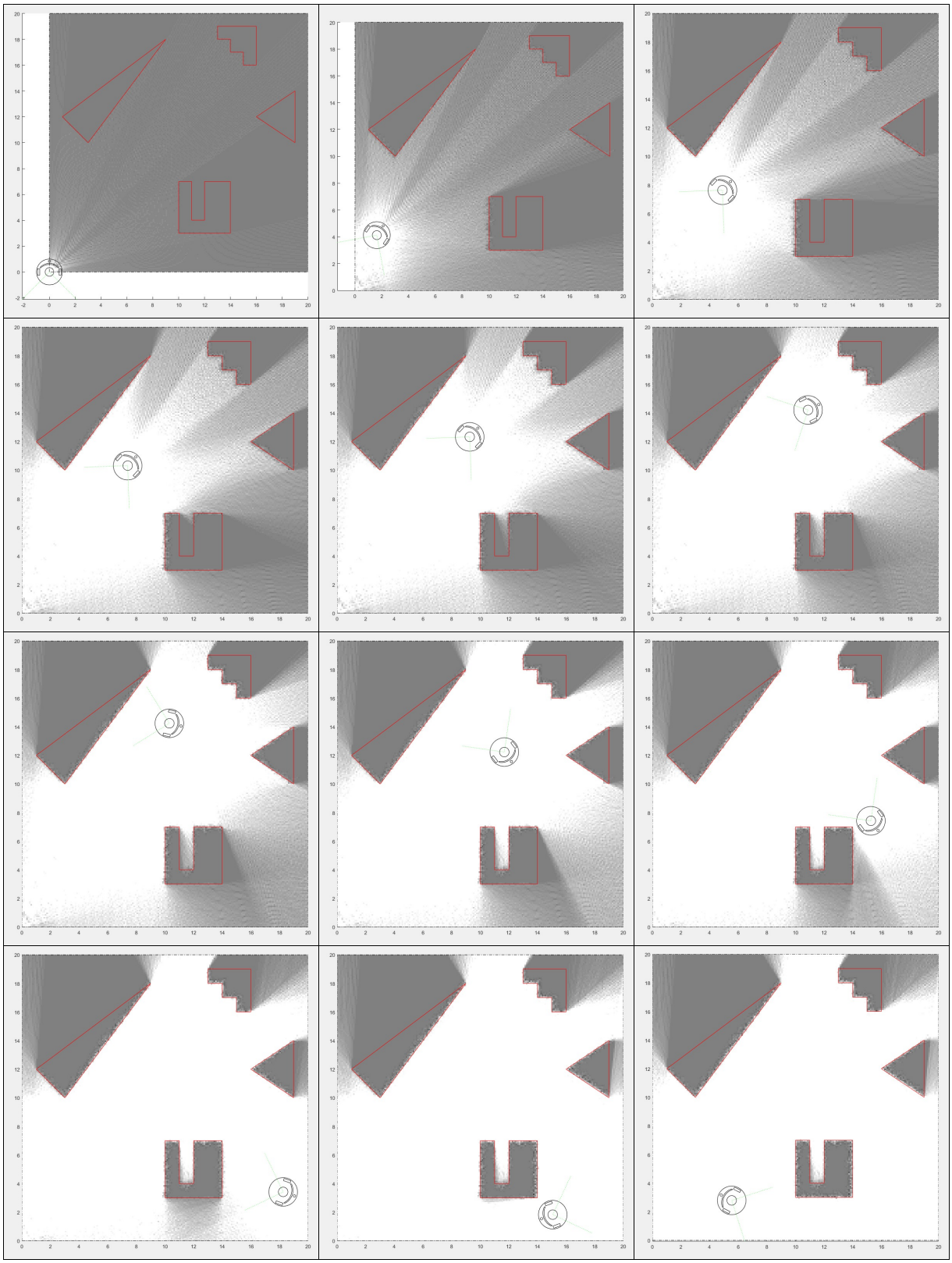


Figura A.41: Resultado de *MappingTB_v7_2* con error de distancia 0.2. Obstáculos en rojo, mapa en escala de grises y pose en negro.

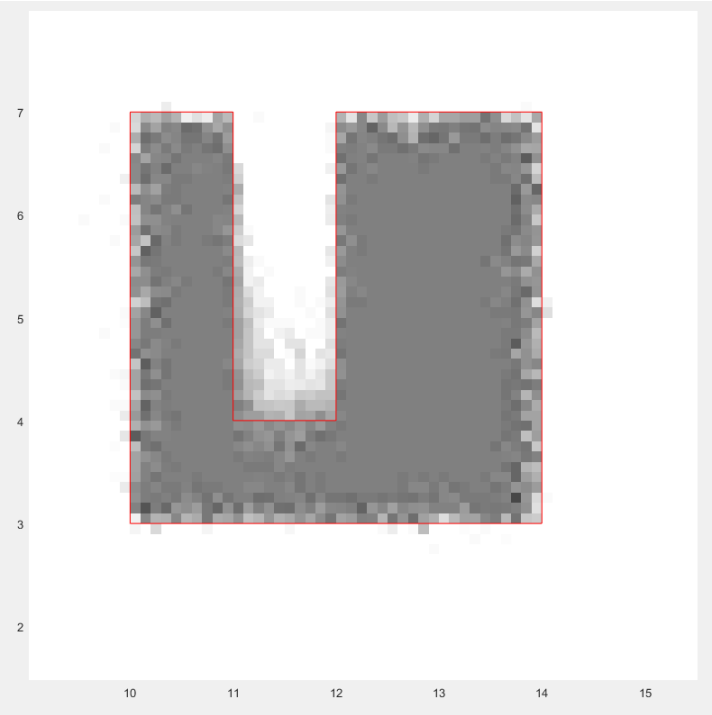


Figura A.42: Ampliación del resultado final de *MappingTB_v7_2* con error de distancia 0.2

Figura A.43: Mapa resultante de *MappingTB_v7_2(0,0.2)*

```
MappingTB_v7_2(0.3,0)
```

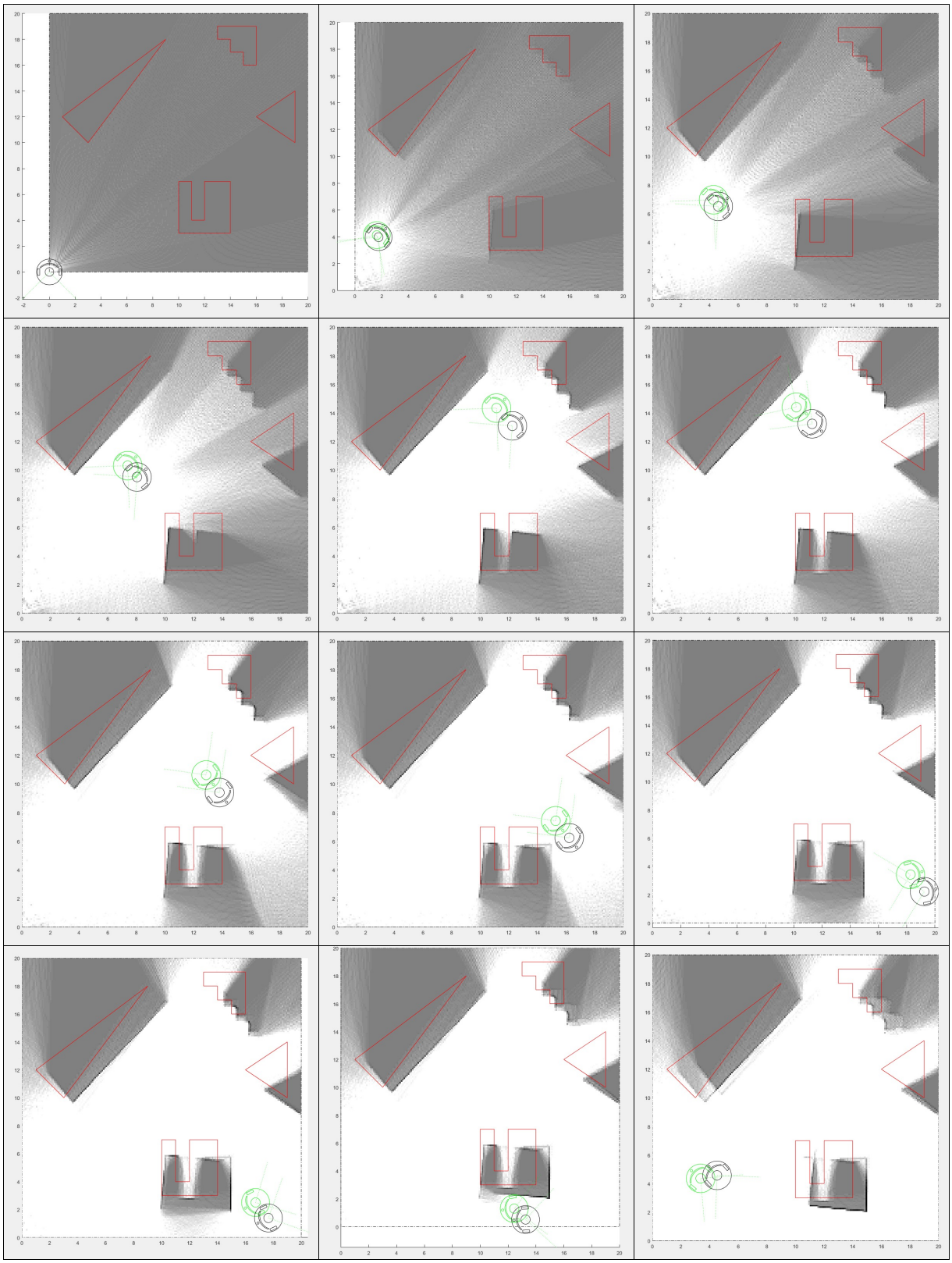


Figura A.44: Resultado de *MappingTB_v7_2* con error de odometría 0.3. Obstáculos en rojo, mapa en escala de grises, pose real en verde y pose calculada en negro.

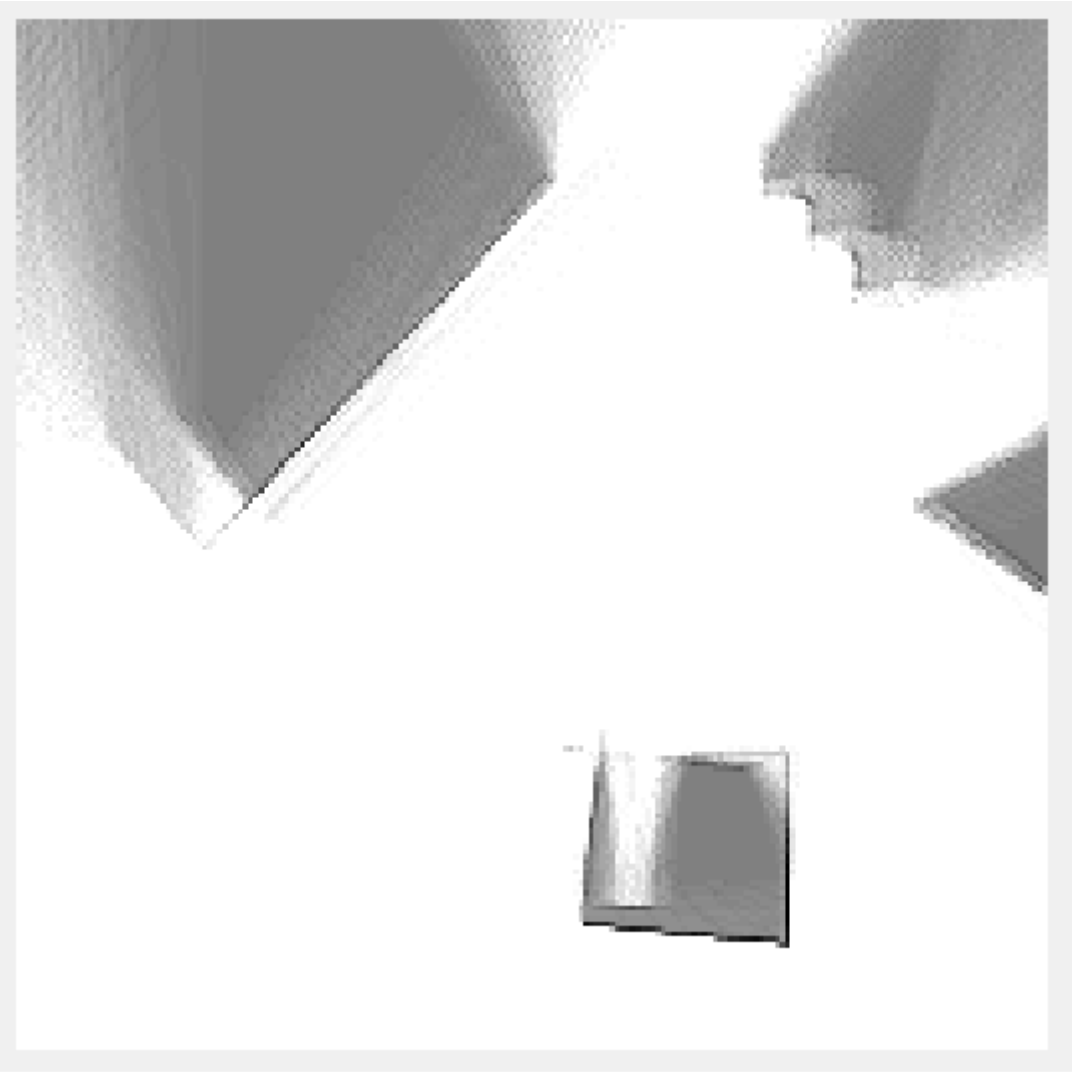


Figura A.45: Mapa resultante de *MappingTB_v7_2(0.3,0)*

A.13. MappingTB_v7_3

```
1 MappingTB_v7_3(0,0)
```

Figura A.46: Resultado de *MappingTB_v7_3* sin error. Obstáculos en rojo, mapa en escala de grises y pose en negro.

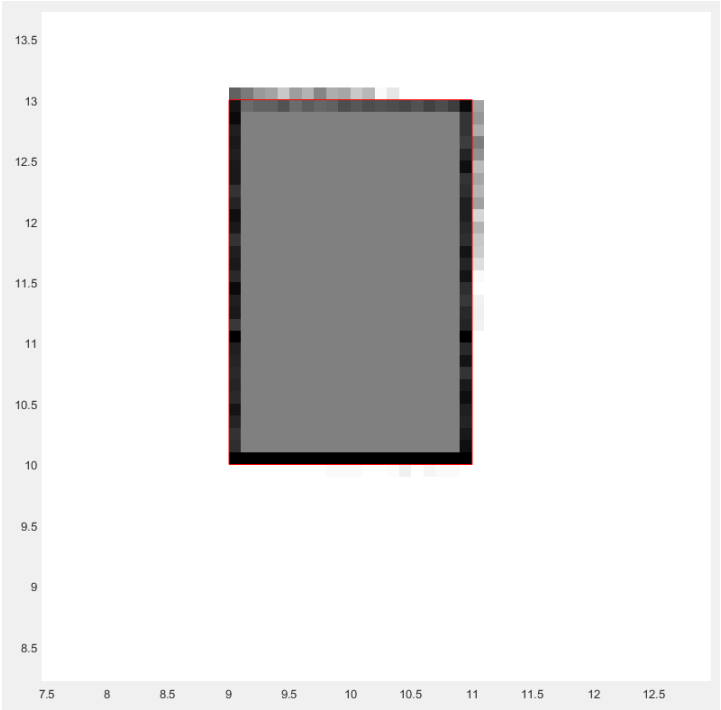


Figura A.47: Ampliación del resultado final de *MappingTB_v7_3* sin error

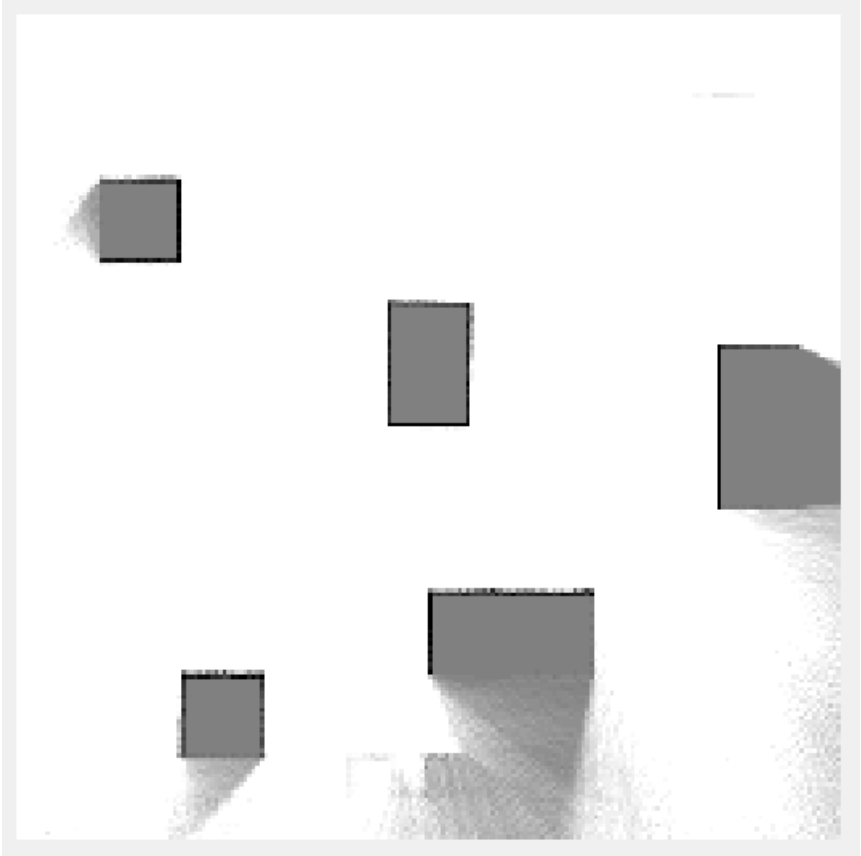


Figura A.48: Mapa resultante de *MappingTB_v7_3(0,0)*

```
1 MappingTB_v7_3(0,0.1)
```

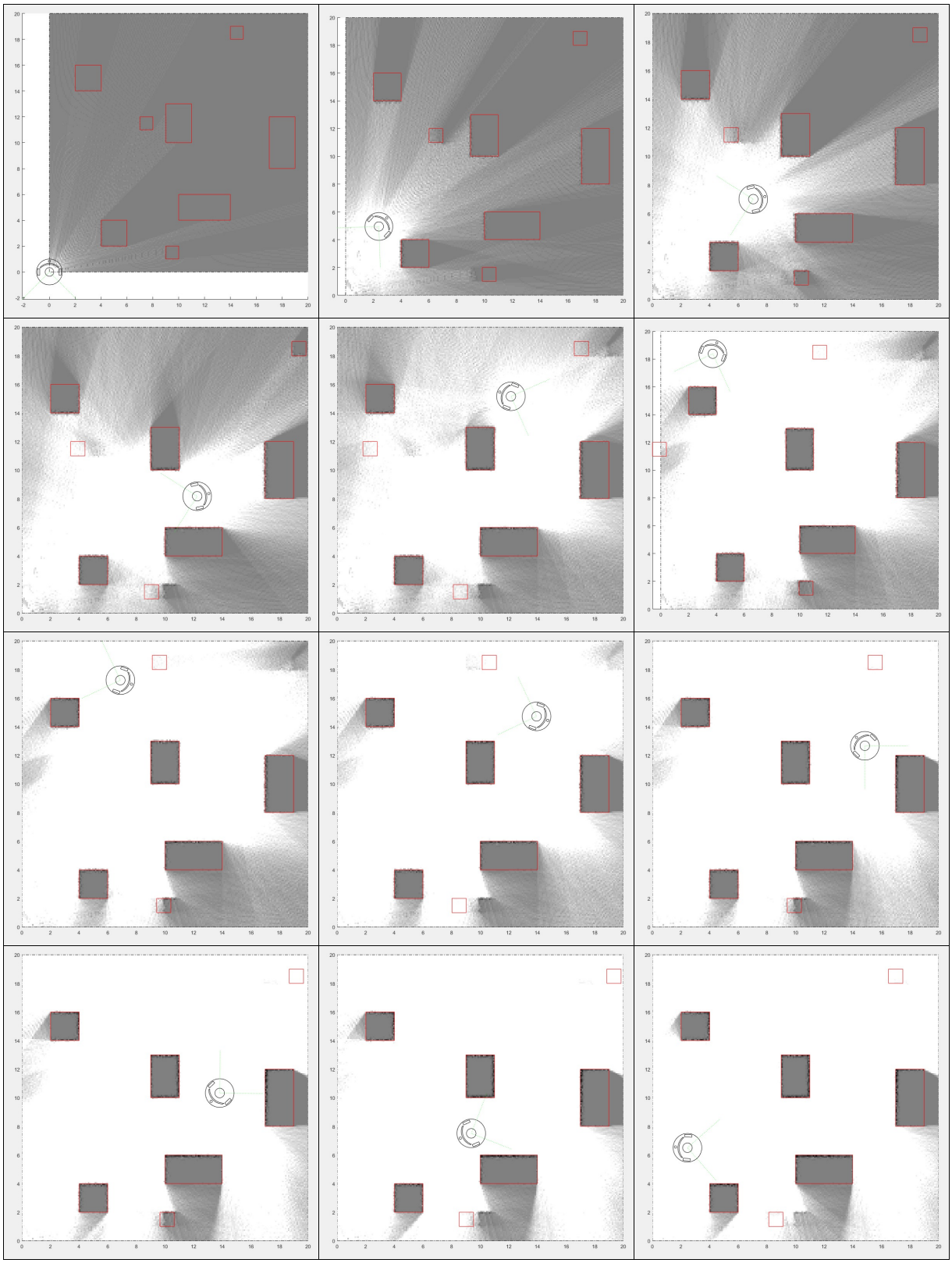


Figura A.49: Resultado de *MappingTB_v7_3* con error de distancia 0.1. Obstáculos en rojo, mapa en escala de grises y pose en negro.

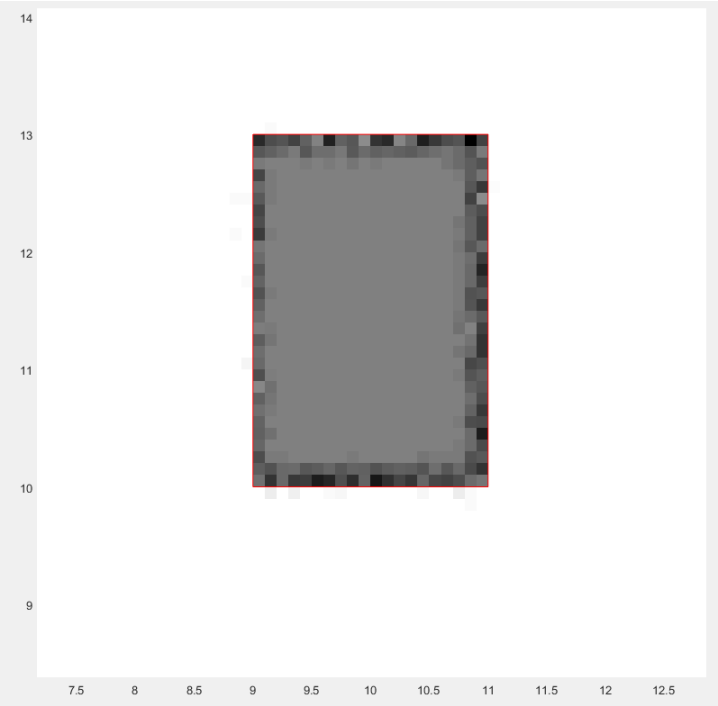


Figura A.50: Ampliación del resultado final de *MappingTB_v7_3* con error de distancia 0.1

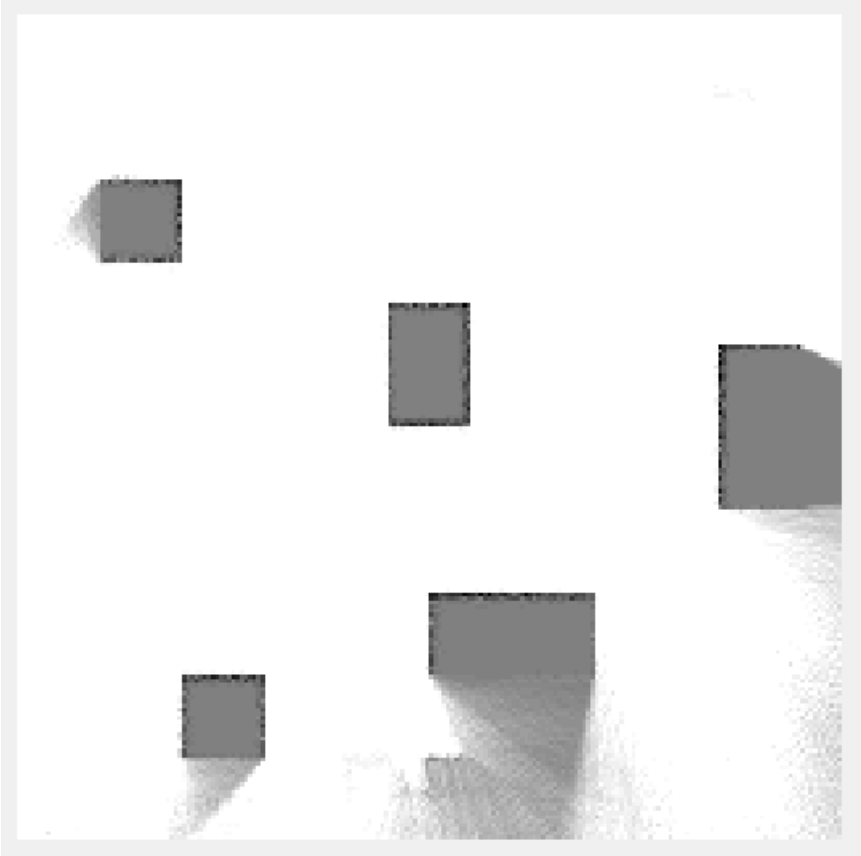


Figura A.51: Mapa resultante de *MappingTB_v7_3 (0,0.1)*

A.14. MappingTB_v8_1

```
1 MappingTB_v8_1(0.5,0)
```

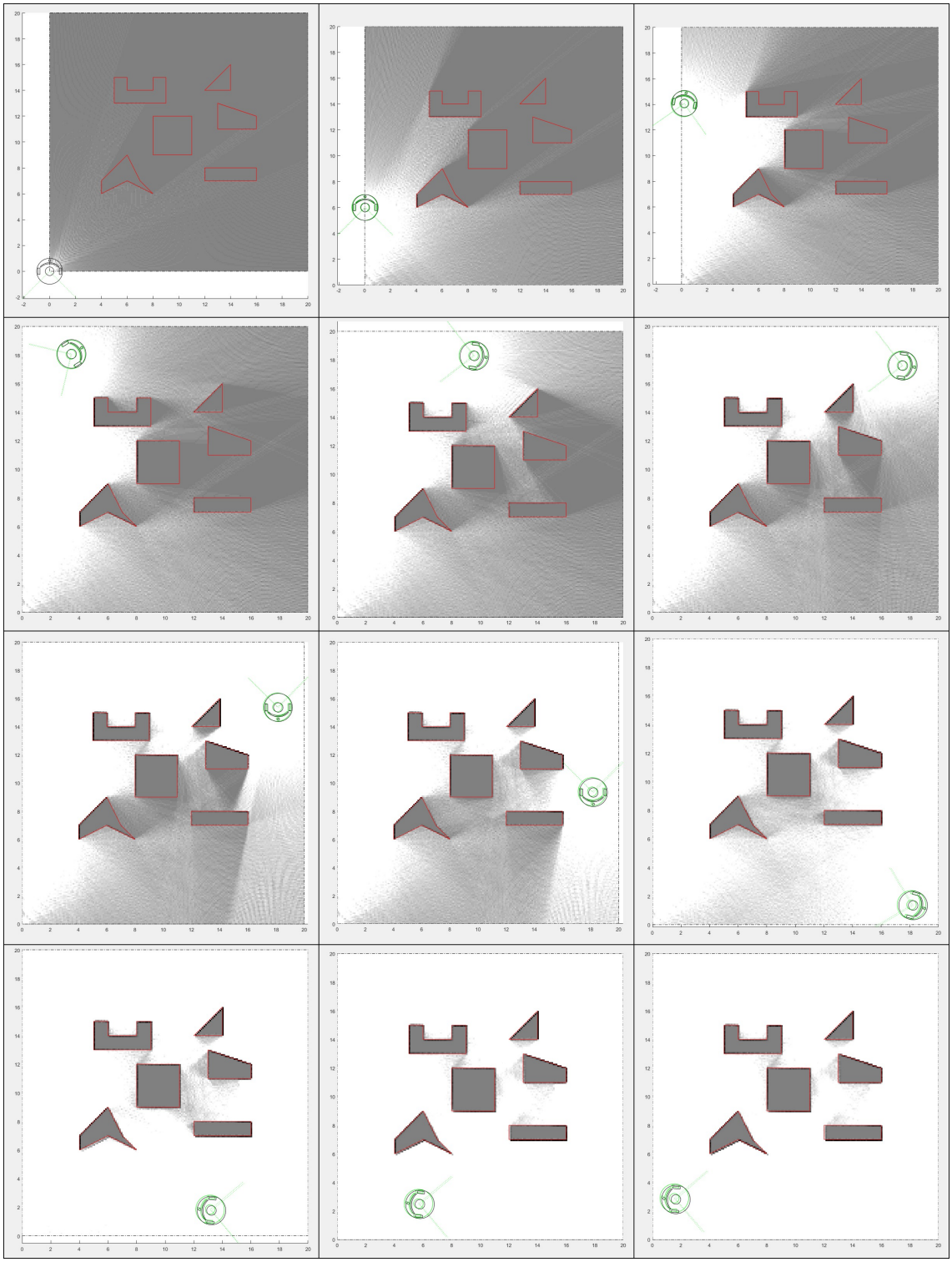


Figura A.52: Resultado de *MappingTB_v8_1* con error de odometría 0.5. Obstáculos en rojo, mapa en escala de grises, pose real en verde y pose calculada en negro.

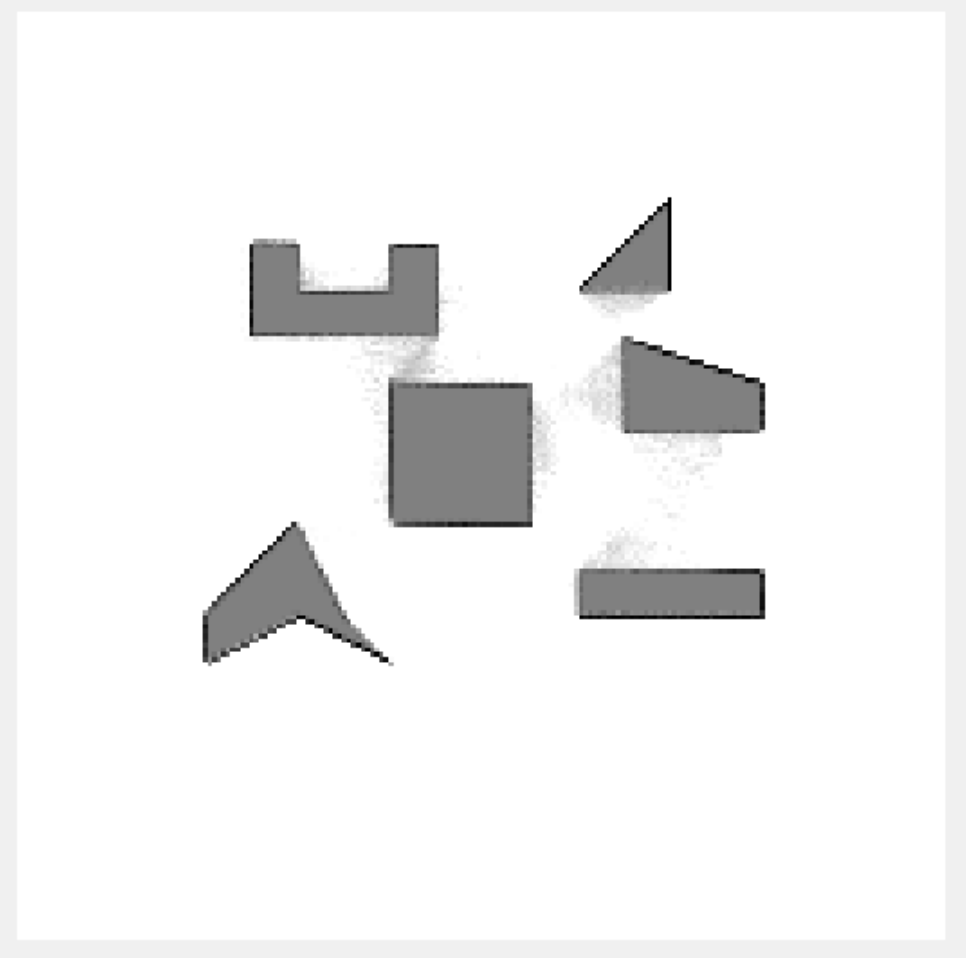


Figura A.53: Mapa resultante de *MappingTB_v8_1(0.5,0)*

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0	1	0	0.0186
3	0	2	0	0.035
4	0	3	0	0.0544
5	0	4	0	0.0535
6	0	5	0	0.039
7	0	6	0	0.0422
8	0	7	0	0.041
9	0	8	0	0.041
10	0	9	0	0.0441
11	0	10	0	0.038
12	0	11	0	0.0382
13	0	12	0	0.0332
14	0	13	0	0.0447
15	0.179	14.04	-0.17	0.0606
16	0.532	15.04	-0.34	0.0571
17	1.049	15.96	-0.51	0.066

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
18	1.714	16.79	-0.68	0.0488
19	2.509	17.49	-0.85	0.0542
20	3.411	18.04	-1.02	0.0594
21	4.394	18.43	-1.19	0.073
22	5.428	18.66	-1.36	0.0777
23	6.486	18.7	-1.53	0.0745
24	7.535	18.56	-1.7	0.0788
25	8.527	18.43	-1.7	0.0733
26	9.519	18.3	-1.7	0.0793
27	10.51	18.18	-1.7	0.0882
28	11.5	18.05	-1.7	0.0909
29	12.49	17.92	-1.7	0.0828
30	13.48	17.79	-1.7	0.0748
31	14.48	17.66	-1.7	0.0784
32	15.47	17.53	-1.7	0.0811
33	16.46	17.4	-1.7	0.0822
34	17.45	17.27	-1.7	0.0692
35	17.59	17.23	-1.88	0.0636
36	17.73	17.16	-2.06	0.0636
37	17.84	17.06	-2.24	0.061
38	17.94	16.95	-2.42	0.058
39	18.02	16.82	-2.6	0.0559
40	18.07	16.68	-2.78	0.0667
41	18.1	16.54	-2.96	0.0644
42	18.1	16.39	-3.14	0.0678
43	18.1	15.39	-3.14	0.0713
44	18.11	14.39	-3.14	0.0745
45	18.11	13.39	-3.14	0.0737
46	18.11	12.39	-3.14	0.0725
47	18.11	11.39	-3.14	0.0735
48	18.11	10.39	-3.14	0.0868
49	18.11	9.385	-3.14	0.1047
50	18.11	8.385	-3.14	0.1091
51	18.12	7.385	-3.14	0.1062
52	18.12	6.385	-3.14	0.1113
53	18.12	5.385	-3.14	0.0994
54	18.12	4.385	-3.14	0.1148
55	18.12	3.385	-3.14	0.122
56	18.12	2.385	-3.14	0.1282
57	18.13	1.385	-3.14	0.1163
58	18.13	1.385	-2.48	0.117
59	18.13	1.385	-1.82	0.121
60	18.13	1.385	-1.16	0.1204
61	18.13	1.385	-0.5	0.1217
62	18.13	1.385	0.16	0.1184

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
63	18.13	1.385	0.82	0.115
64	18.13	1.385	1.48	0.1198
65	17.3	1.46	1.48	0.1132
66	16.47	1.536	1.48	0.1232
67	15.64	1.611	1.48	0.1218
68	14.81	1.687	1.48	0.1345
69	13.98	1.763	1.48	0.1266
70	13.15	1.838	1.48	0.1181
71	12.32	1.914	1.48	0.1228
72	11.49	1.989	1.48	0.1151
73	10.66	2.065	1.48	0.1151
74	9.827	2.14	1.48	0.1092
75	8.997	2.216	1.48	0.1053
76	8.167	2.291	1.48	0.1143
77	7.337	2.367	1.48	0.1172
78	6.507	2.443	1.48	0.1139
79	5.677	2.518	1.48	0.117
80	4.847	2.594	1.48	0.1109
81	4.017	2.669	1.48	0.114
82	3.188	2.745	1.48	0.123
83	2.358	2.82	1.48	0.1376
84	1.528	2.896	1.48	0.1442
Error cuadrático medio				0.0853

Tabla A.8: Error de pose en *MappingTB_v8_1*(0.5,0)

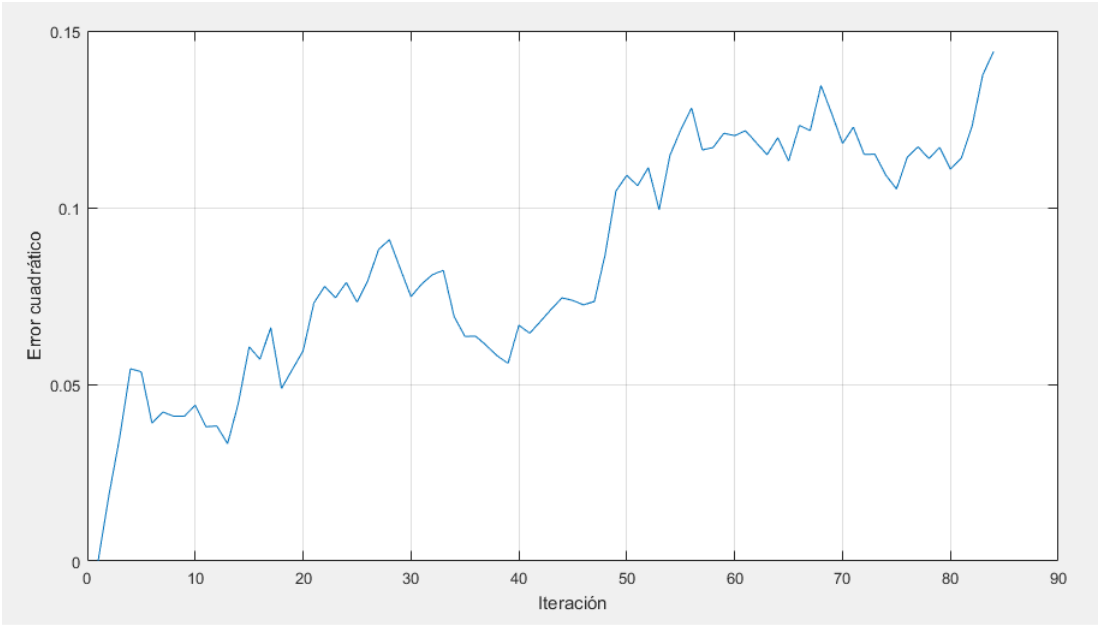


Figura A.54: Gráfica del error de pose en *MappingTB_v8_1*(0.5,0)

```
MappingTB_v8_1(0.8,0)
```

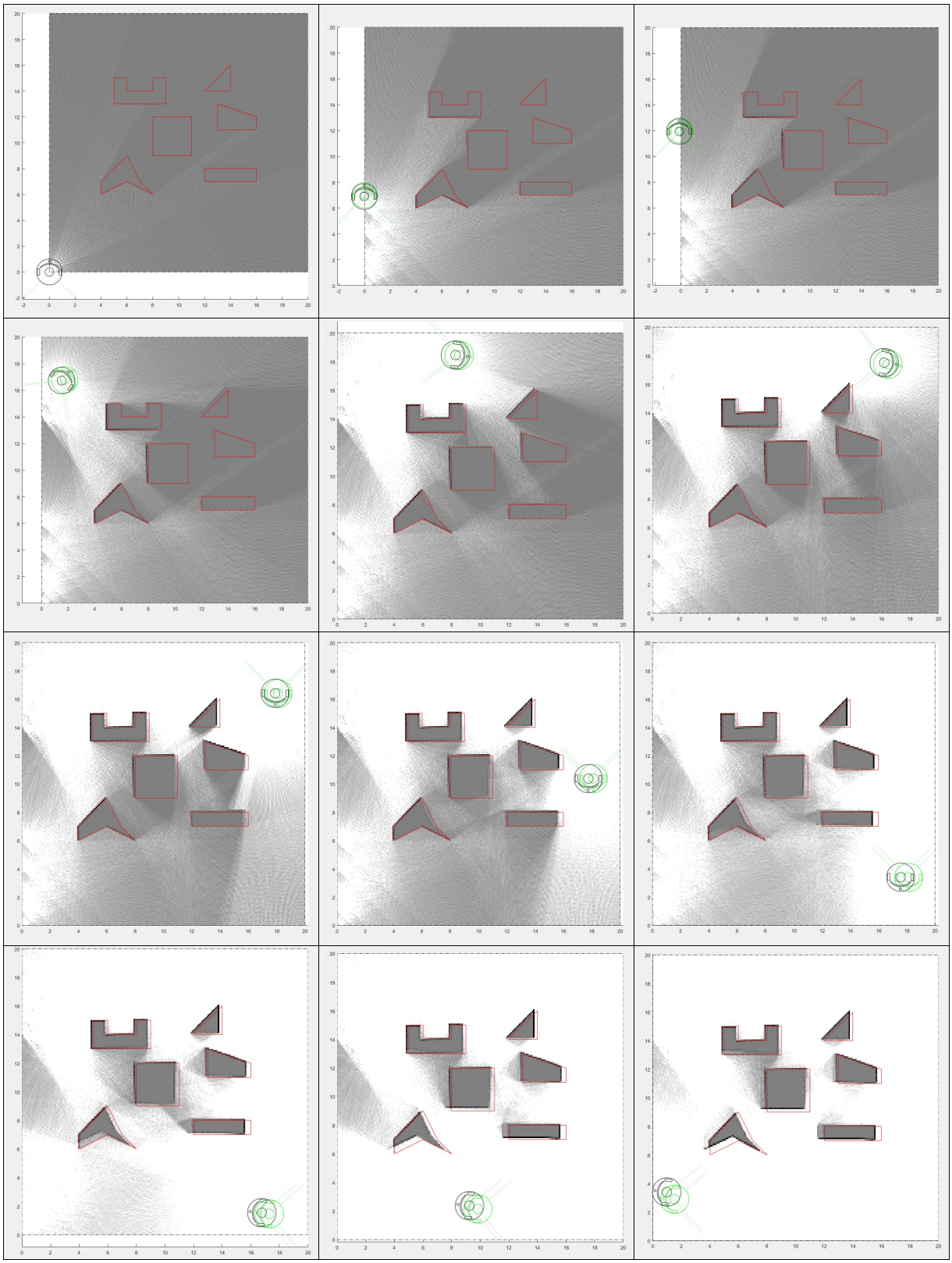


Figura A.55: Resultado de *MappingTB_v8_1* con error de odometría 0.8. Obstáculos en rojo, mapa en escala de grises, pose real en verde y pose calculada en negro.

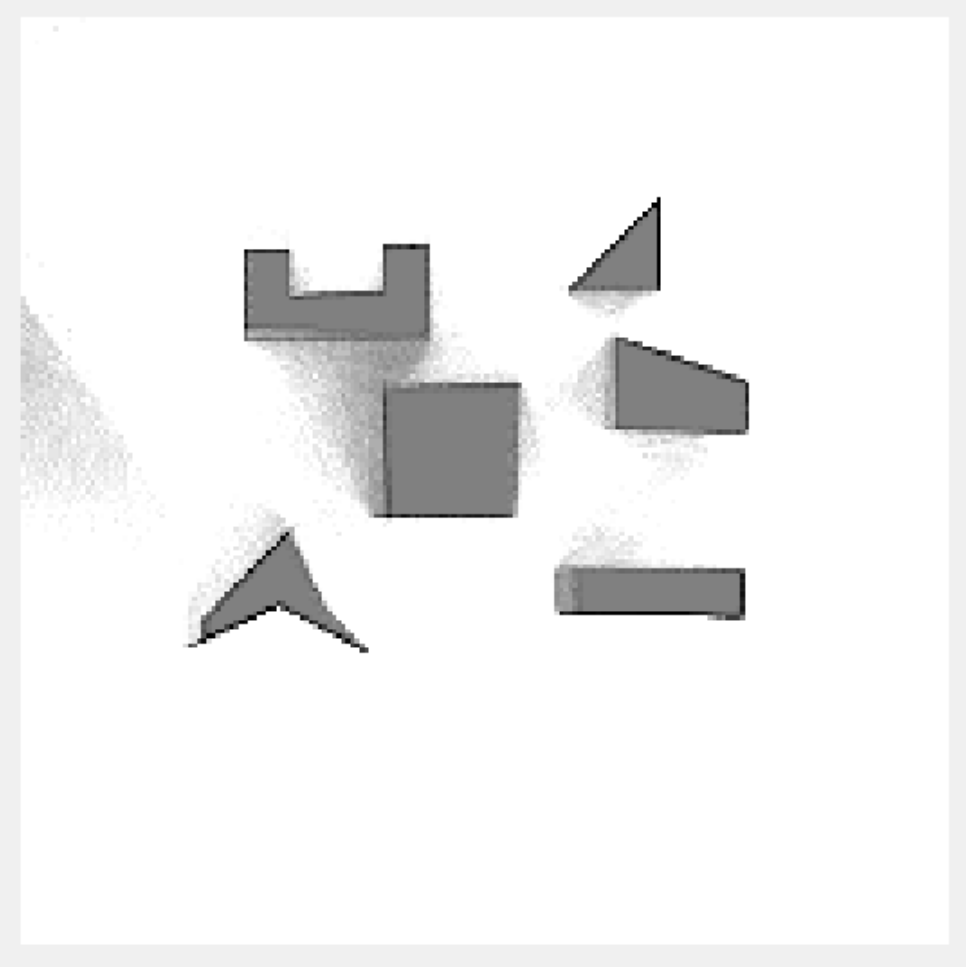


Figura A.56: Mapa resultante de *MappingTB_v8_1(0.8,0)*

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0	1	0	0.0307
3	0	2	0	0.0249
4	0	3	0	0.0282
5	0	4	0	0.0491
6	0	5	0	0.0687
7	0	6	0	0.0795
8	0	7	0	0.1122
9	0	8	0	0.1218
10	0	9	0	0.1171
11	0	10	0	0.1301
12	0	11	0	0.1373
13	0	12	0	0.1552
14	0	13	0	0.1618
15	0.179	14.04	-0.17	0.1411
16	0.532	15.04	-0.34	0.1808

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
17	1.049	15.96	-0.51	0.2207
18	1.714	16.79	-0.68	0.2335
19	2.509	17.49	-0.85	0.2535
20	3.411	18.04	-1.02	0.231
21	4.394	18.43	-1.19	0.2448
22	5.428	18.66	-1.36	0.2369
23	6.486	18.7	-1.53	0.2453
24	7.535	18.56	-1.7	0.2414
25	8.527	18.43	-1.7	0.2522
26	9.519	18.3	-1.7	0.2581
27	10.51	18.18	-1.7	0.2706
28	11.5	18.05	-1.7	0.274
29	12.49	17.92	-1.7	0.2678
30	13.48	17.79	-1.7	0.2933
31	14.48	17.66	-1.7	0.2995
32	15.47	17.53	-1.7	0.2888
33	16.46	17.4	-1.7	0.2789
34	17.45	17.27	-1.7	0.2691
35	17.59	17.23	-1.88	0.2634
36	17.73	17.16	-2.06	0.2352
37	17.84	17.06	-2.24	0.232
38	17.94	16.95	-2.42	0.2009
39	18.02	16.82	-2.6	0.2027
40	18.07	16.68	-2.78	0.2066
41	18.1	16.54	-2.96	0.2073
42	18.1	16.39	-3.14	0.204
43	18.1	15.39	-3.14	0.2339
44	18.11	14.39	-3.14	0.2652
45	18.11	13.39	-3.14	0.2948
46	18.11	12.39	-3.14	0.3007
47	18.11	11.39	-3.14	0.3274
48	18.11	10.39	-3.14	0.3474
49	18.11	9.385	-3.14	0.3862
50	18.11	8.385	-3.14	0.4082
51	18.12	7.385	-3.14	0.4181
52	18.12	6.385	-3.14	0.4563
53	18.12	5.385	-3.14	0.4795
54	18.12	4.385	-3.14	0.5189
55	18.12	3.385	-3.14	0.5598
56	18.12	2.385	-3.14	0.5636
57	18.13	1.385	-3.14	0.5526
58	18.13	1.385	-2.48	0.5436
59	18.13	1.385	-1.82	0.5727
60	18.13	1.385	-1.16	0.5672
61	18.13	1.385	-0.5	0.5808

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
62	18.13	1.385	0.16	0.5843
63	18.13	1.385	0.82	0.5695
64	18.13	1.385	1.48	0.5515
65	17.3	1.46	1.48	0.531
66	16.47	1.536	1.48	0.5473
67	15.64	1.611	1.48	0.5506
68	14.81	1.687	1.48	0.5731
69	13.98	1.763	1.48	0.5773
70	13.15	1.838	1.48	0.5678
71	12.32	1.914	1.48	0.5887
72	11.49	1.989	1.48	0.5994
73	10.66	2.065	1.48	0.609
74	9.827	2.14	1.48	0.6183
75	8.997	2.216	1.48	0.6249
76	8.167	2.291	1.48	0.6162
77	7.337	2.367	1.48	0.6238
78	6.507	2.443	1.48	0.63
79	5.677	2.518	1.48	0.6469
80	4.847	2.594	1.48	0.6585
81	4.017	2.669	1.48	0.6515
82	3.188	2.745	1.48	0.688
83	2.358	2.82	1.48	0.7275
84	1.528	2.896	1.48	0.7474
Error cuadrático medio				0.362

Tabla A.9: Error de pose en *MappingTB_v8_1*(0.8,0)

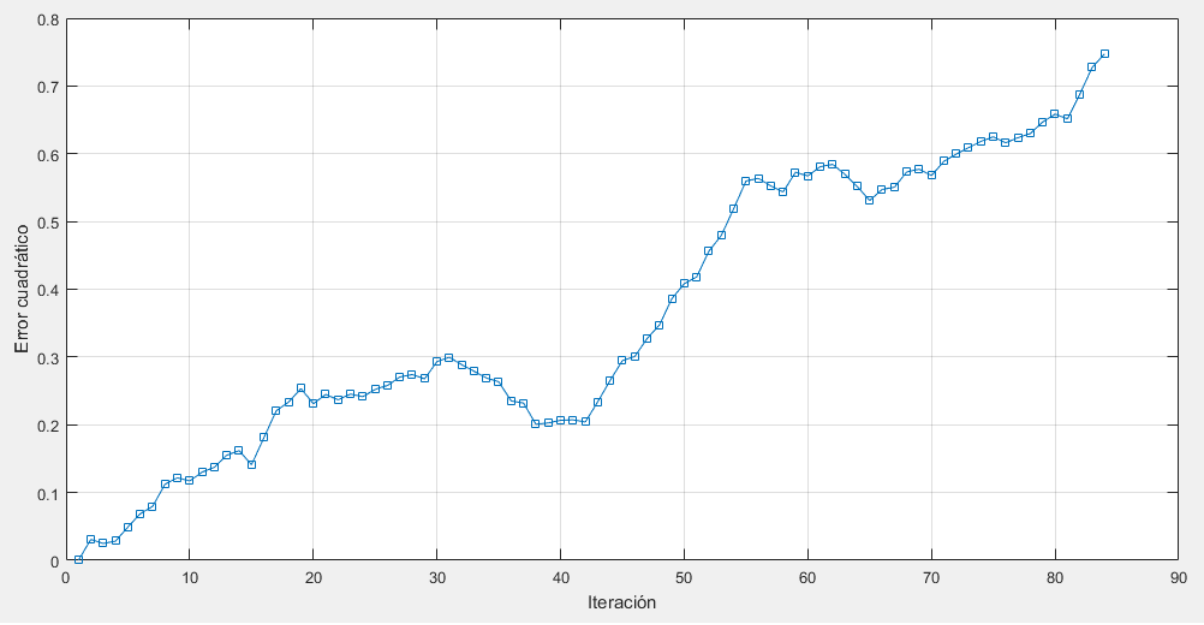


Figura A.57: Gráfica del error de pose en *MappingTB_v8_1*(0.8,0)

Profile Summary

Generated 05-Sep-2016 01:46:26 using performance time.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
MappingTB_v8_1	1	50.995 s	8.508 s	
VissionSimulation_v1	84	34.833 s	0.712 s	
Mapping_v7	84	5.032 s	0.300 s	
CalculateMAP_v3	83	4.444 s	3.033 s	
mapshow	84	1.124 s	0.019 s	
dibujar_v2	168	0.566 s	0.479 s	

Figura A.58: Desempeño temporal de *MappingTB_v8_1(0.8,0)*

Se han realizado varias simulaciones de *MappingTB_v8_1(0.8,0)* para tener una idea de cuantas veces se corrige adecuadamente la posición. A continuación los resultados.

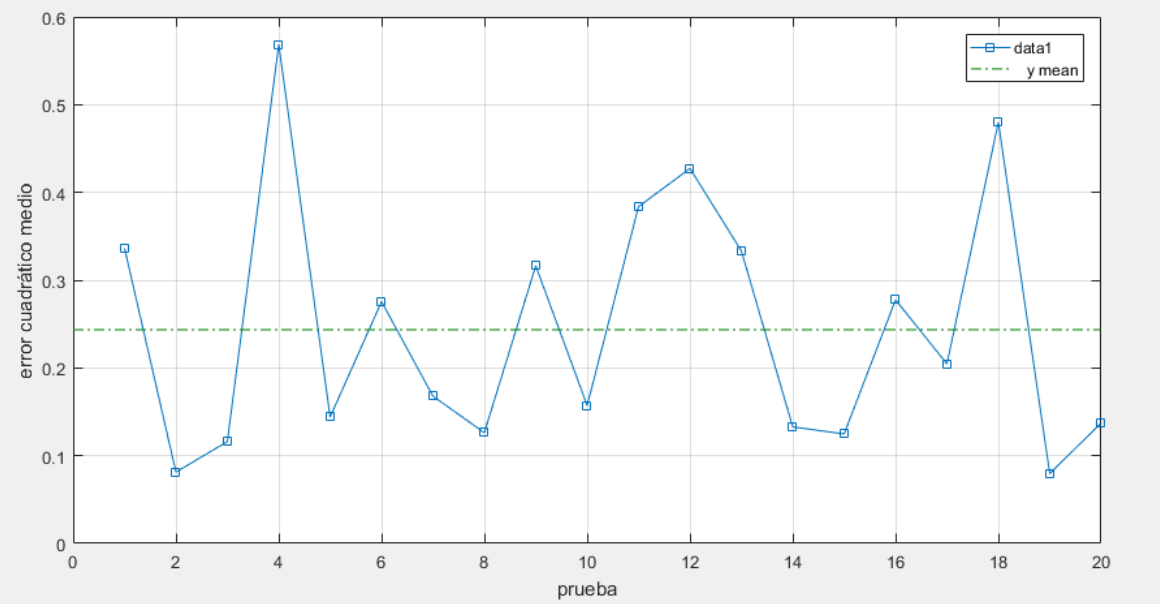


Figura A.59: Gráfica de errores cuadráticos medios para múltiples simulaciones de *MappingTB_v8_1(0.8,0)*

```
1 MappingTB_v8_1(1.1,0)
```

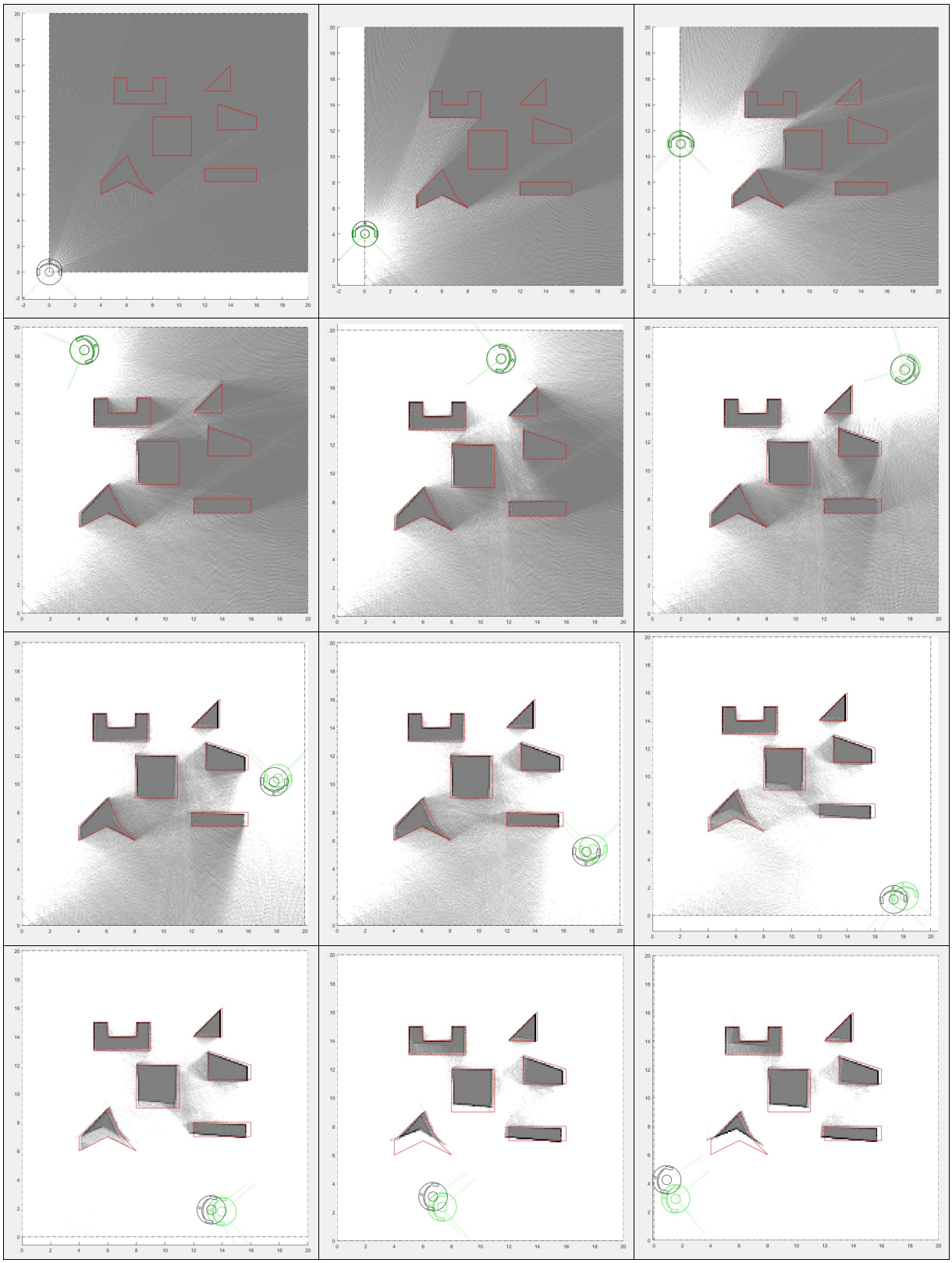


Figura A.60: Resultado de *MappingTB_v8_1* con error de odometría 1.1. Obstáculos en rojo, mapa en escala de grises, pose real en verde y pose calculada en negro.

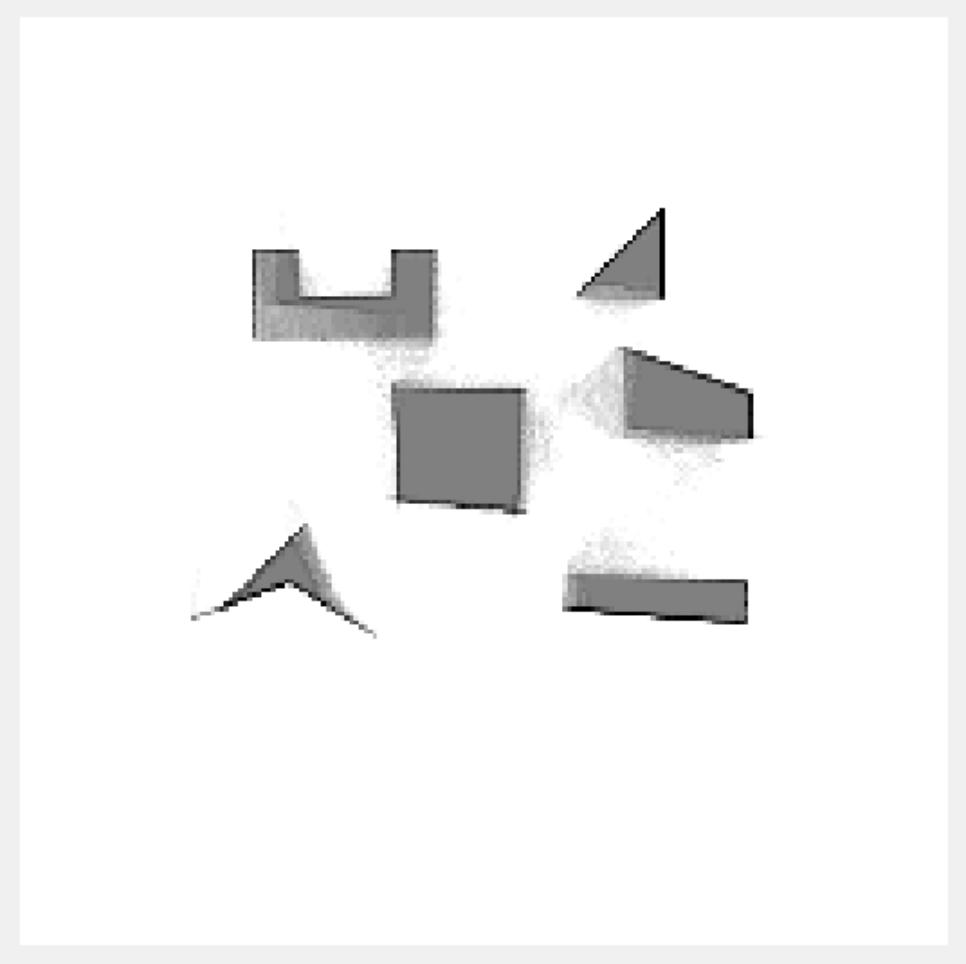


Figura A.61: Mapa resultante de *MappingTB_v8_1(1.1,0)*

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0	1	0	0.022
3	0	2	0	0.0609
4	0	3	0	0.0337
5	0	4	0	0.0373
6	0	5	0	0.0799
7	0	6	0	0.0987
8	0	7	0	0.1515
9	0	8	0	0.1578
10	0	9	0	0.1619
11	0	10	0	0.144
12	0	11	0	0.1183
13	0	12	0	0.0911
14	0	13	0	0.08
15	0.179	14.04	-0.17	0.1039
16	0.532	15.04	-0.34	0.067

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
17	1.049	15.96	-0.51	0.0621
18	1.714	16.79	-0.68	0.071
19	2.509	17.49	-0.85	0.0526
20	3.411	18.04	-1.02	0.0468
21	4.394	18.43	-1.19	0.0651
22	5.428	18.66	-1.36	0.0506
23	6.486	18.7	-1.53	0.0924
24	7.535	18.56	-1.7	0.0499
25	8.527	18.43	-1.7	0.0387
26	9.519	18.3	-1.7	0.0394
27	10.51	18.18	-1.7	0.0622
28	11.5	18.05	-1.7	0.0892
29	12.49	17.92	-1.7	0.0954
30	13.48	17.79	-1.7	0.1116
31	14.48	17.66	-1.7	0.1184
32	15.47	17.53	-1.7	0.1407
33	16.46	17.4	-1.7	0.1464
34	17.45	17.27	-1.7	0.158
35	17.59	17.23	-1.88	0.1735
36	17.73	17.16	-2.06	0.1722
37	17.84	17.06	-2.24	0.1747
38	17.94	16.95	-2.42	0.1627
39	18.02	16.82	-2.6	0.1665
40	18.07	16.68	-2.78	0.1886
41	18.1	16.54	-2.96	0.2015
42	18.1	16.39	-3.14	0.2277
43	18.1	15.39	-3.14	0.2454
44	18.11	14.39	-3.14	0.2514
45	18.11	13.39	-3.14	0.2654
46	18.11	12.39	-3.14	0.2741
47	18.11	11.39	-3.14	0.314
48	18.11	10.39	-3.14	0.3638
49	18.11	9.385	-3.14	0.402
50	18.11	8.385	-3.14	0.4233
51	18.12	7.385	-3.14	0.4201
52	18.12	6.385	-3.14	0.4621
53	18.12	5.385	-3.14	0.528
54	18.12	4.385	-3.14	0.5735
55	18.12	3.385	-3.14	0.6255
56	18.12	2.385	-3.14	0.7094
57	18.13	1.385	-3.14	0.798
58	18.13	1.385	-2.48	0.7948
59	18.13	1.385	-1.82	0.8202
60	18.13	1.385	-1.16	0.8193
61	18.13	1.385	-0.5	0.8284

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
62	18.13	1.385	0.16	0.8362
63	18.13	1.385	0.82	0.8457
64	18.13	1.385	1.48	0.8289
65	17.3	1.46	1.48	0.8154
66	16.47	1.536	1.48	0.8199
67	15.64	1.611	1.48	0.8033
68	14.81	1.687	1.48	0.7515
69	13.98	1.763	1.48	0.7484
70	13.15	1.838	1.48	0.7546
71	12.32	1.914	1.48	0.7729
72	11.49	1.989	1.48	0.8004
73	10.66	2.065	1.48	0.8372
74	9.827	2.14	1.48	0.8531
75	8.997	2.216	1.48	0.8989
76	8.167	2.291	1.48	0.9339
77	7.337	2.367	1.48	0.9659
78	6.507	2.443	1.48	0.9941
79	5.677	2.518	1.48	1.072
80	4.847	2.594	1.48	1.143
81	4.017	2.669	1.48	1.233
82	3.188	2.745	1.48	1.316
83	2.358	2.82	1.48	1.399
84	1.528	2.896	1.48	1.477
Error cuadrático medio				0.4308

Tabla A.10: Error de pose en *MappingTB_v8_1*(1.1,0)

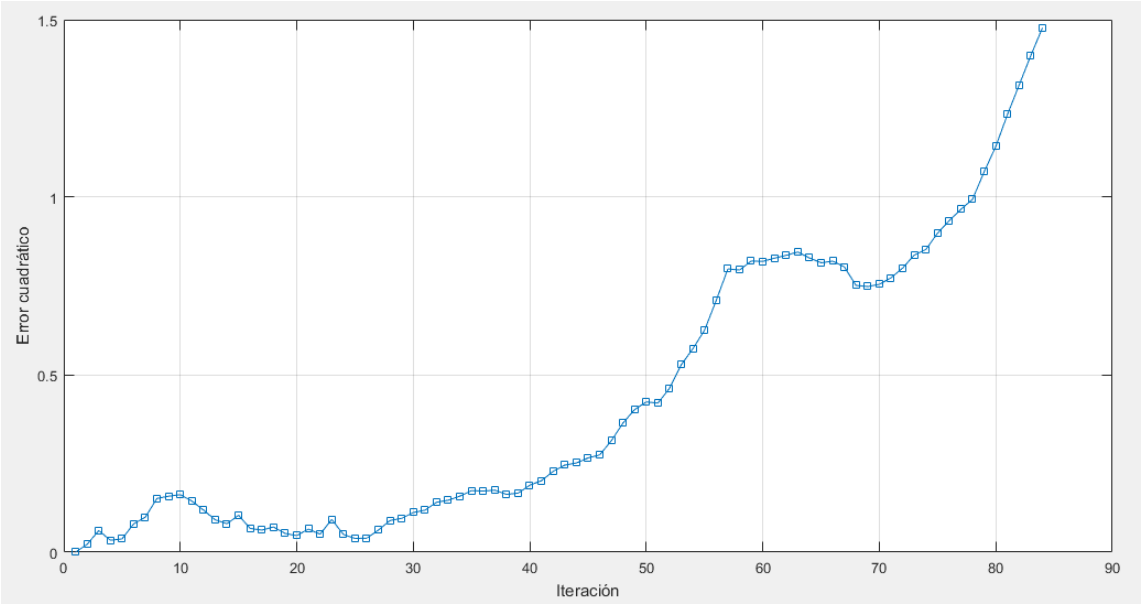


Figura A.62: Gráfica del error de pose en *MappingTB_v8_1*(1.1,0)

A.15. MappingTB_v8_2

```
1 MappingTB_v8_2(0.16,0)
```

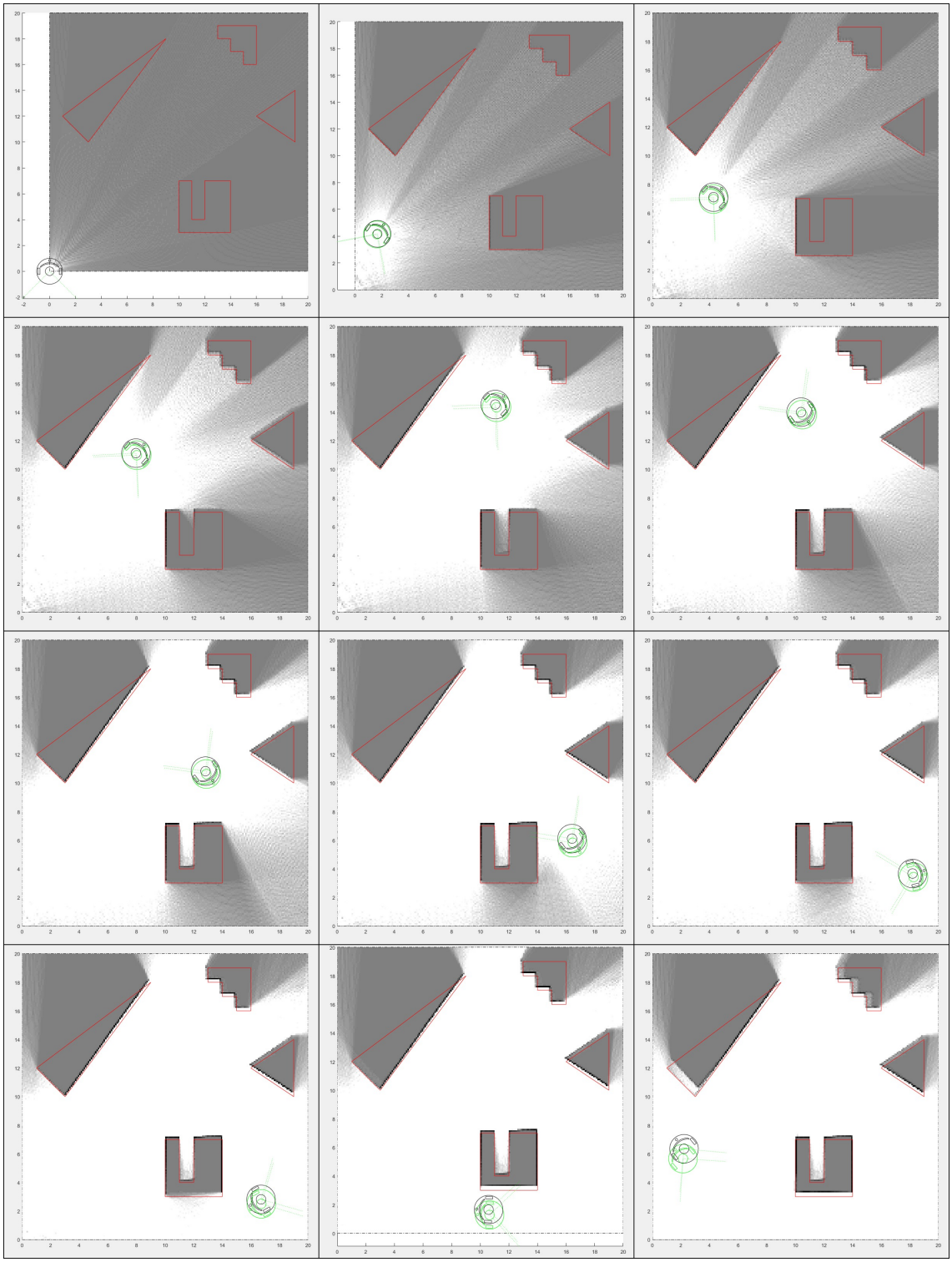


Figura A.63: Resultado de *MappingTB_v8_2* con error de odometría 0.16. Obstáculos en rojo, mapa en escala de grises, pose real en verde y pose calculada en negro.

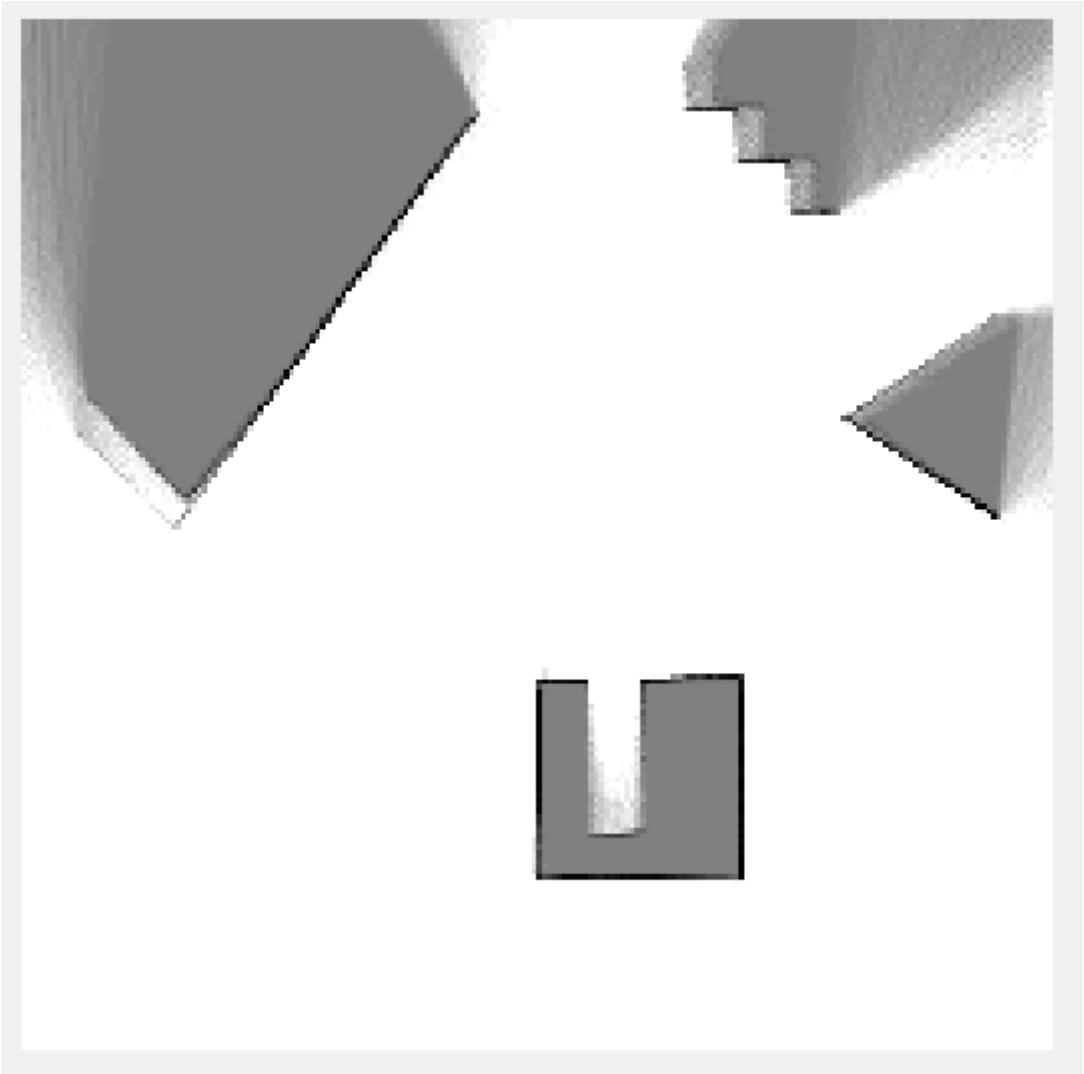


Figura A.64: Mapa resultante de *MappingTB_v8_1(0.16,0)*

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0.168	1.112	-0.15	0.0092
3	0.501	2.187	-0.3	0.0365
4	0.99	3.2	-0.45	0.0264
5	1.625	4.129	-0.6	0.0666
6	2.392	4.952	-0.75	0.072
7	3.017	5.622	-0.75	0.0817
8	3.642	6.293	-0.75	0.1118
9	4.266	6.964	-0.75	0.1395
10	4.891	7.635	-0.75	0.1408
11	5.516	8.305	-0.75	0.1437
12	6.141	8.976	-0.75	0.1311
13	6.766	9.647	-0.75	0.1514
14	7.391	10.32	-0.75	0.1618

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
15	8.016	10.99	-0.75	0.1615
16	8.64	11.66	-0.75	0.1552
17	9.265	12.33	-0.75	0.1783
18	9.89	13	-0.75	0.2014
19	10.52	13.67	-0.75	0.2248
20	11.14	14.34	-0.75	0.2142
21	10.88	14.21	-1.1	0.2194
22	10.59	14.17	-1.45	0.2212
23	10.31	14.24	-1.8	0.2064
24	10.06	14.4	-2.15	0.2029
25	9.888	14.63	-2.5	0.1929
26	10.49	13.83	-2.5	0.1948
27	11.08	13.03	-2.5	0.2095
28	11.68	12.23	-2.5	0.2045
29	12.28	11.43	-2.5	0.2034
30	12.88	10.63	-2.5	0.2099
31	13.48	9.827	-2.5	0.2144
32	14.08	9.026	-2.5	0.2385
33	14.68	8.225	-2.5	0.2604
34	15.27	7.424	-2.5	0.2754
35	15.87	6.622	-2.5	0.2898
36	16.47	5.821	-2.5	0.2931
37	17.07	5.02	-2.5	0.2832
38	17.67	4.219	-2.5	0.2697
39	18.27	3.418	-2.5	0.2574
40	18.27	3.418	-1.91	0.2624
41	18.27	3.418	-1.32	0.2599
42	18.27	3.418	-0.73	0.2709
43	18.27	3.418	-0.14	0.2812
44	18.27	3.418	0.45	0.3074
45	18.27	3.418	1.04	0.3132
46	18.27	3.418	1.63	0.3117
47	18.27	3.418	2.22	0.3242
48	17.51	2.931	2.145	0.2927
49	16.73	2.502	2.07	0.3138
50	15.91	2.134	1.995	0.3013
51	15.07	1.827	1.92	0.2943
52	14.21	1.585	1.845	0.2973
53	13.33	1.407	1.77	0.3241
54	12.44	1.296	1.695	0.3386
55	11.55	1.252	1.62	0.3647
56	10.65	1.275	1.545	0.3768
57	9.758	1.366	1.47	0.4117
58	8.876	1.522	1.395	0.4498
59	8.008	1.745	1.32	0.4894

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
60	7.16	2.031	1.245	0.5261
61	6.335	2.381	1.17	0.5594
62	5.539	2.791	1.095	0.5813
63	4.775	3.26	1.02	0.6031
64	4.049	3.785	0.945	0.6169
65	3.364	4.362	0.87	0.6516
66	2.725	4.99	0.795	0.6551
67	2.134	5.663	0.72	0.694
Error cuadrático medio				0.2735

Tabla A.11: Error de pose en *MappingTB_v8_2*(0.16,0)

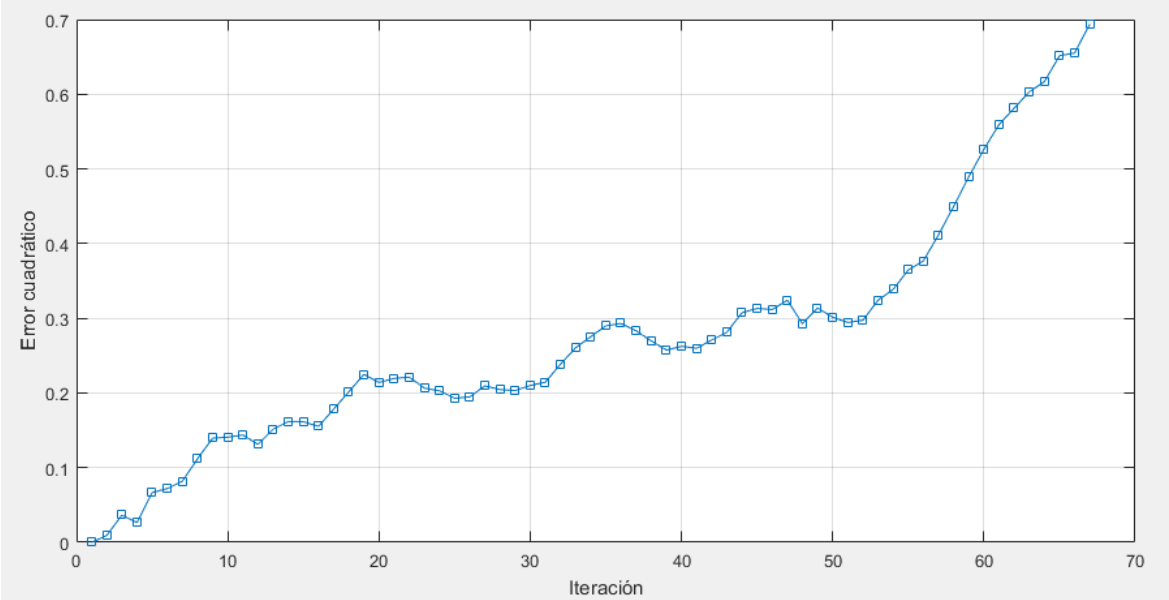


Figura A.65: Gráfica del error de pose en *MappingTB_v8_2*(0.16,0)

```
1 MappingTB_v8_2(0.3,0)
```

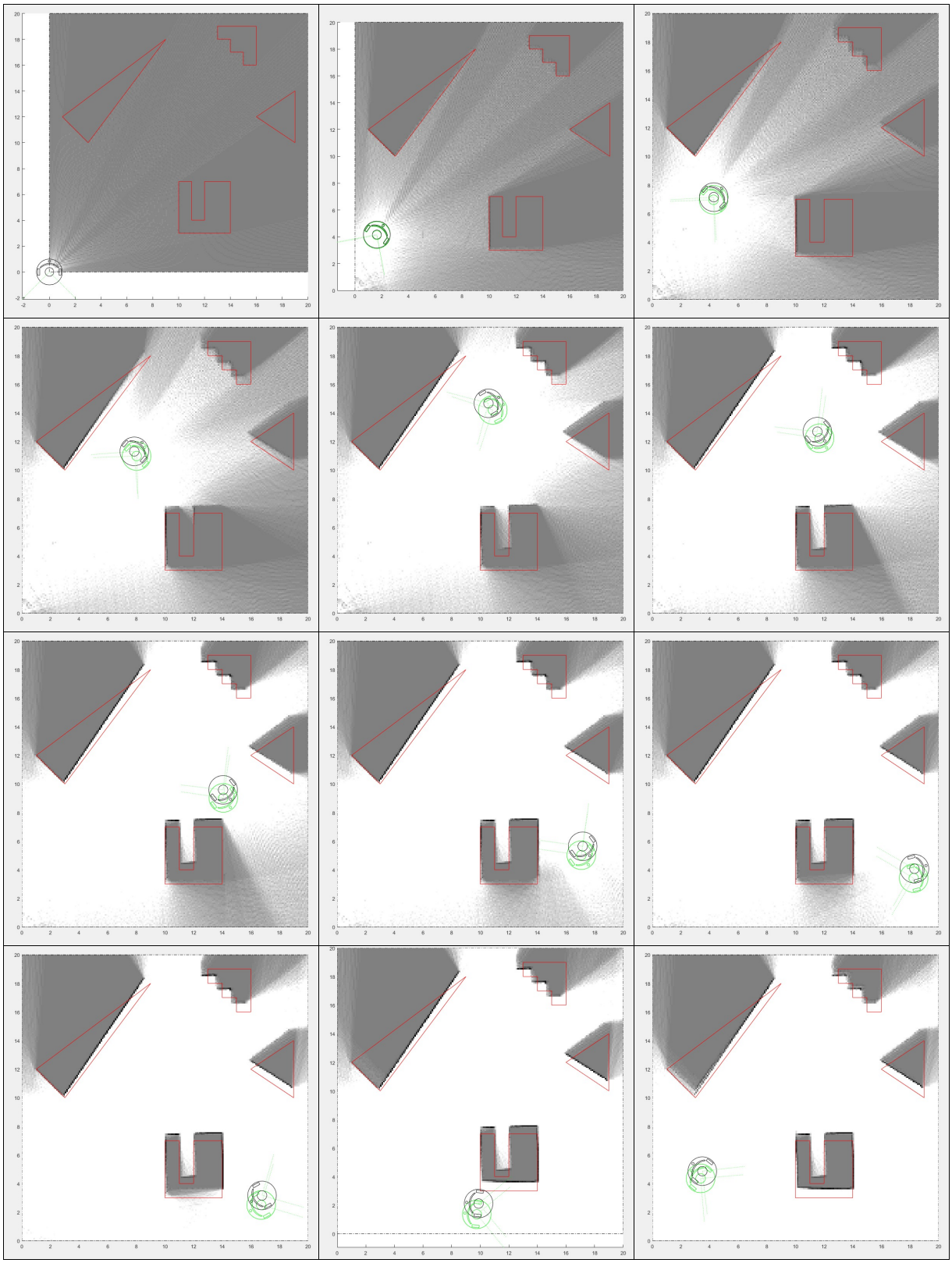


Figura A.66: Resultado de *MappingTB_v8_2* con error de odometría 0.3. Obstáculos en rojo, mapa en escala de grises, pose real en verde y pose calculada en negro.

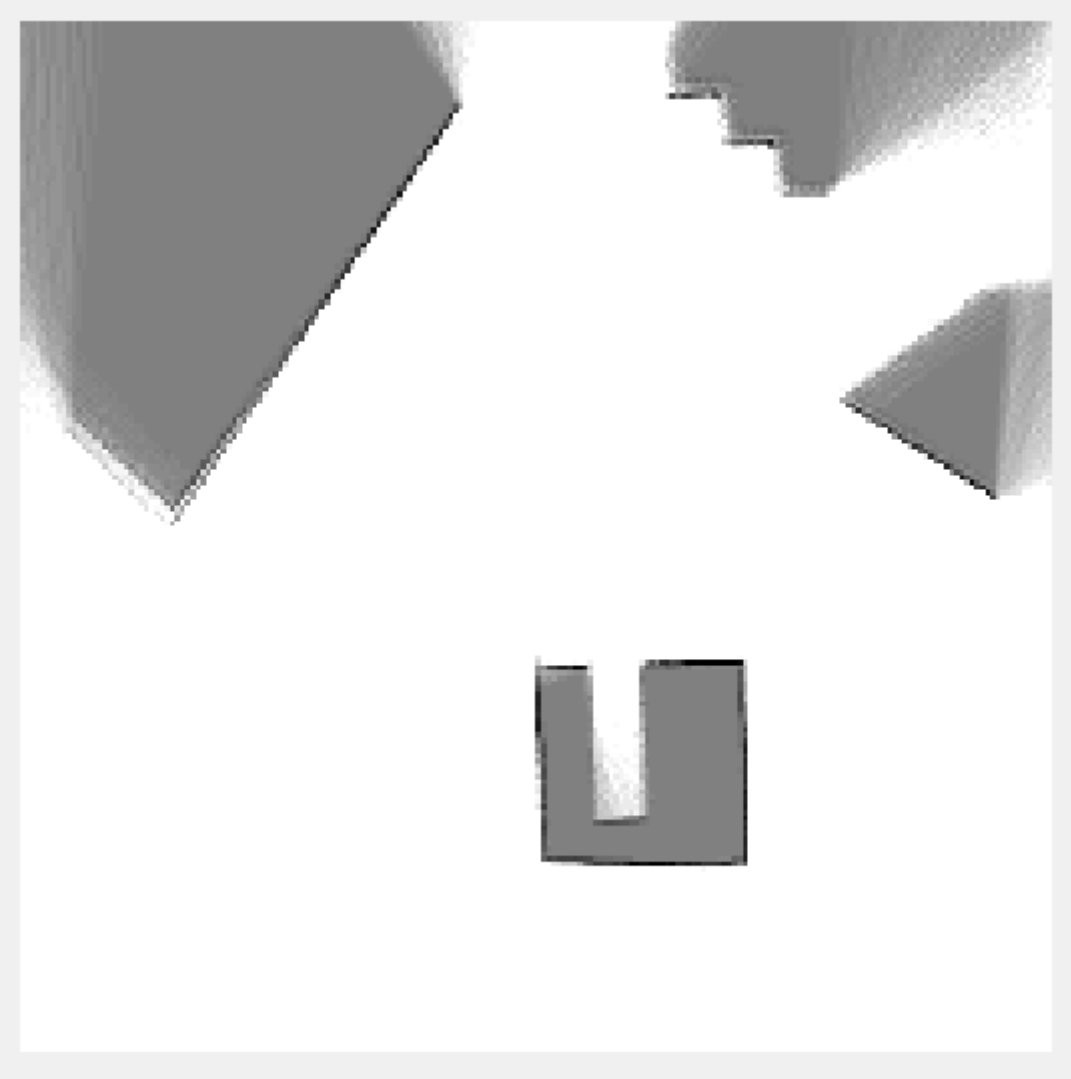


Figura A.67: Mapa resultante de *MappingTB_v8_1(0.3,0)*

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0.168	1.112	-0.15	0.0495
3	0.501	2.187	-0.3	0.0733
4	0.99	3.2	-0.45	0.0411
5	1.625	4.129	-0.6	0.0459
6	2.392	4.952	-0.75	0.1031
7	3.017	5.622	-0.75	0.1156
8	3.642	6.293	-0.75	0.1746
9	4.266	6.964	-0.75	0.1876
10	4.891	7.635	-0.75	0.2243
11	5.516	8.305	-0.75	0.2695
12	6.141	8.976	-0.75	0.2737
13	6.766	9.647	-0.75	0.3059
14	7.391	10.32	-0.75	0.3312

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
15	8.016	10.99	-0.75	0.3675
16	8.64	11.66	-0.75	0.4139
17	9.265	12.33	-0.75	0.4556
18	9.89	13	-0.75	0.5194
19	10.52	13.67	-0.75	0.5251
20	11.14	14.34	-0.75	0.5693
21	10.88	14.21	-1.1	0.5586
22	10.59	14.17	-1.45	0.5647
23	10.31	14.24	-1.8	0.5478
24	10.06	14.4	-2.15	0.5274
25	9.888	14.63	-2.5	0.5232
26	10.49	13.83	-2.5	0.5092
27	11.08	13.03	-2.5	0.4736
28	11.68	12.23	-2.5	0.4874
29	12.28	11.43	-2.5	0.5271
30	12.88	10.63	-2.5	0.5334
31	13.48	9.827	-2.5	0.5451
32	14.08	9.026	-2.5	0.5627
33	14.68	8.225	-2.5	0.5769
34	15.27	7.424	-2.5	0.599
35	15.87	6.622	-2.5	0.6073
36	16.47	5.821	-2.5	0.6319
37	17.07	5.02	-2.5	0.6301
38	17.67	4.219	-2.5	0.6244
39	18.27	3.418	-2.5	0.6487
40	18.27	3.418	-1.91	0.6536
41	18.27	3.418	-1.32	0.6589
42	18.27	3.418	-0.73	0.678
43	18.27	3.418	-0.14	0.6846
44	18.27	3.418	0.45	0.7108
45	18.27	3.418	1.04	0.7211
46	18.27	3.418	1.63	0.7228
47	18.27	3.418	2.22	0.7348
48	17.51	2.931	2.145	0.7167
49	16.73	2.502	2.07	0.6712
50	15.91	2.134	1.995	0.7169
51	15.07	1.827	1.92	0.6976
52	14.21	1.585	1.845	0.6808
53	13.33	1.407	1.77	0.6938
54	12.44	1.296	1.695	0.6588
55	11.55	1.252	1.62	0.6586
56	10.65	1.275	1.545	0.7214
57	9.758	1.366	1.47	0.7664
58	8.876	1.522	1.395	0.7902
59	8.008	1.745	1.32	0.761

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
60	7.16	2.031	1.245	0.717
61	6.335	2.381	1.17	0.6749
62	5.539	2.791	1.095	0.6357
63	4.775	3.26	1.02	0.5814
64	4.049	3.785	0.945	0.5604
65	3.364	4.362	0.87	0.5206
66	2.725	4.99	0.795	0.519
67	2.134	5.663	0.72	0.4774
Error cuadrático medio				0.5151

Tabla A.12: Error de pose en *MappingTB_v8_2*(0.3,0)

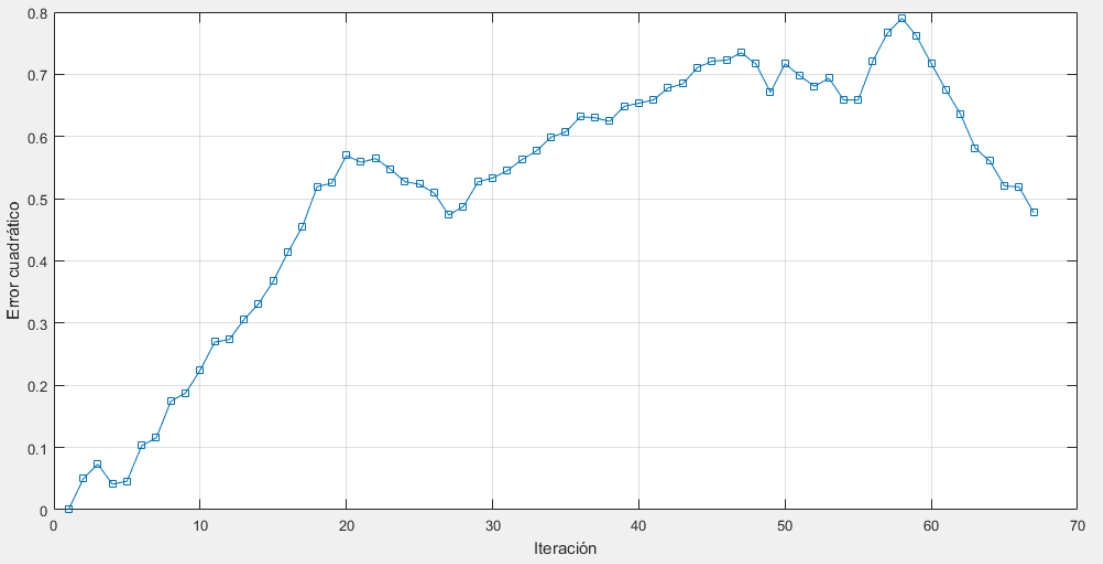


Figura A.68: Gráfica del error de pose en *MappingTB_v8_2*(0.3,0)

A.16. MappingTB_v9_1

```
1 MappingTB_v9_1(0.5,0)
```

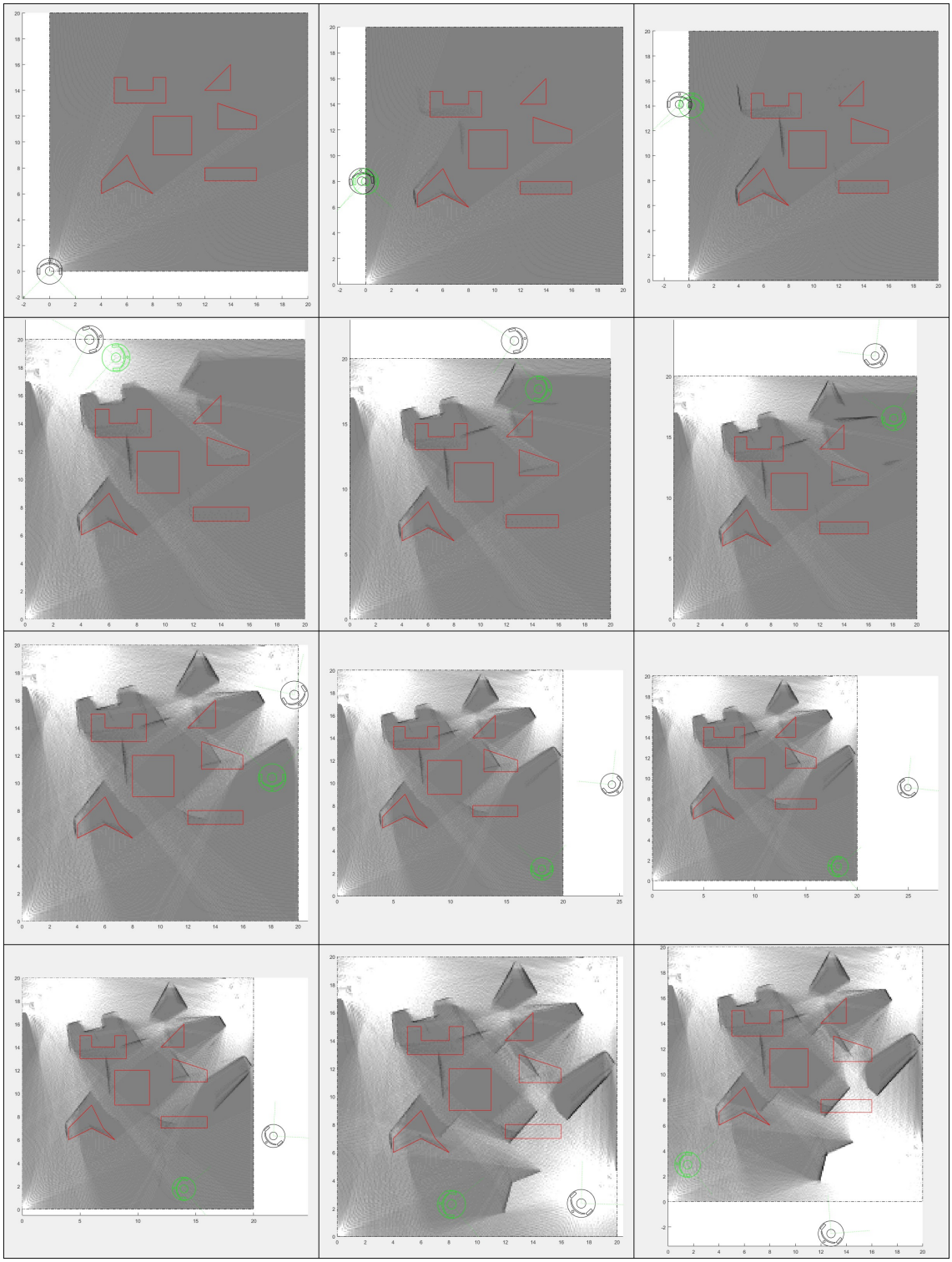


Figura A.69: Resultado de *MappingTB_v9_1* con error de odometría 0.5. Obstáculos en rojo, mapa en escala de grises, pose real en verde y pose calculada en negro.

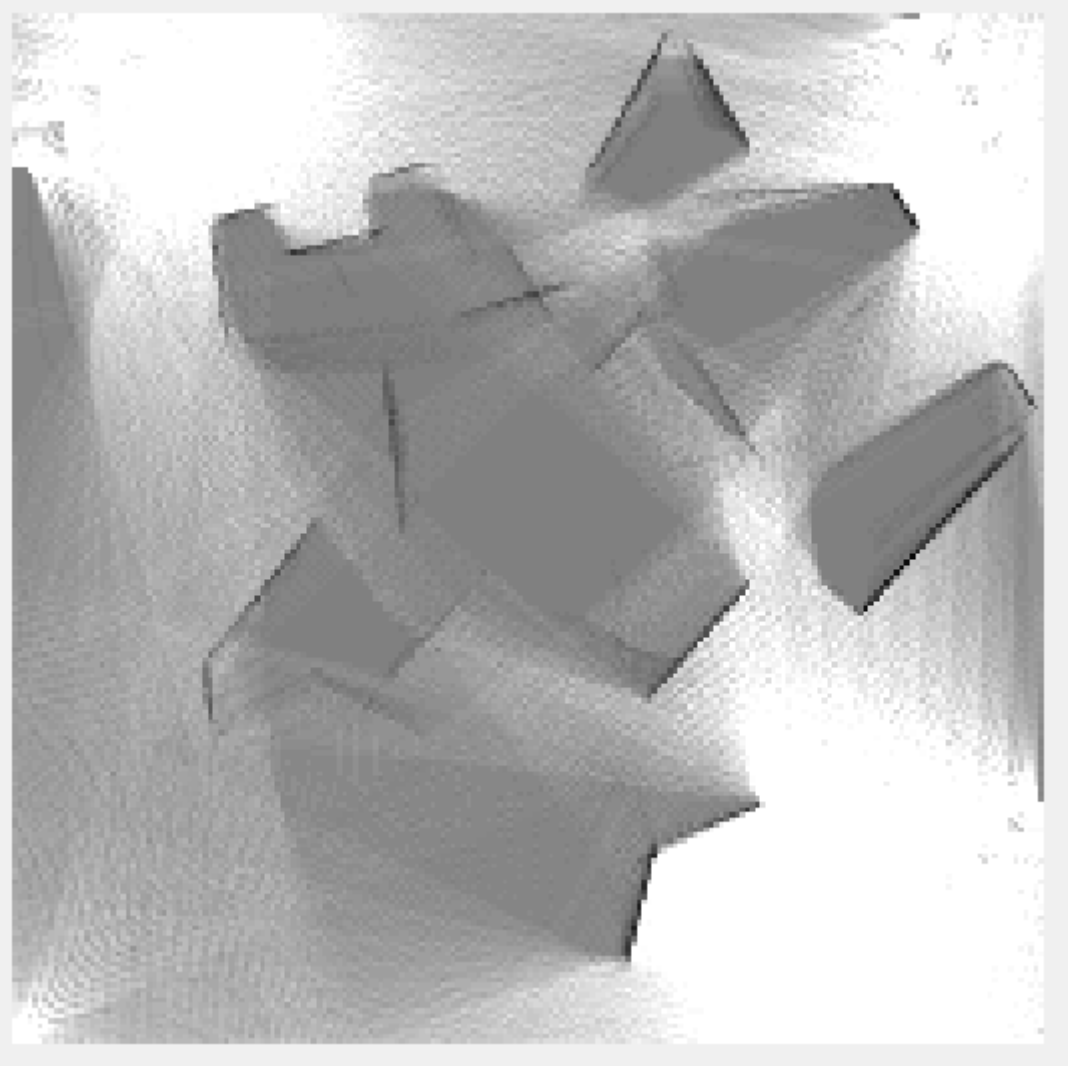


Figura A.70: Mapa resultante de *MappingTB_v9_1(0.5,0)*

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0	1	0	0.0216
3	0	2	0	0.0397
4	0	3	0	0.0547
5	0	4	0	0.0877
6	0	5	0	0.1286
7	0	6	0	0.1811
8	0	7	0	0.246
9	0	8	0	0.3305
10	0	9	0	0.4138
11	0	10	0	0.5095
12	0	11	0	0.5988
13	0	12	0	0.7091
14	0	13	0	0.8264

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
15	0.179	14.04	-0.17	0.9666
16	0.532	15.04	-0.34	1.103
17	1.049	15.96	-0.51	1.252
18	1.714	16.79	-0.68	1.41
19	2.509	17.49	-0.85	1.578
20	3.411	18.04	-1.02	1.755
21	4.394	18.43	-1.19	1.941
22	5.428	18.66	-1.36	2.13
23	6.486	18.7	-1.53	2.311
24	7.535	18.56	-1.7	2.469
25	8.527	18.43	-1.7	2.644
26	9.519	18.3	-1.7	2.833
27	10.51	18.18	-1.7	3.048
28	11.5	18.05	-1.7	3.286
29	12.49	17.92	-1.7	3.554
30	13.48	17.79	-1.7	3.844
31	14.48	17.66	-1.7	4.15
32	15.47	17.53	-1.7	4.477
33	16.46	17.4	-1.7	4.821
34	17.45	17.27	-1.7	5.188
35	17.59	17.23	-1.88	5.238
36	17.73	17.16	-2.06	5.29
37	17.84	17.06	-2.24	5.322
38	17.94	16.95	-2.42	5.355
39	18.02	16.82	-2.6	5.376
40	18.07	16.68	-2.78	5.384
41	18.1	16.54	-2.96	5.382
42	18.1	16.39	-3.14	5.378
43	18.1	15.39	-3.14	5.395
44	18.11	14.39	-3.14	5.471
45	18.11	13.39	-3.14	5.591
46	18.11	12.39	-3.14	5.752
47	18.11	11.39	-3.14	5.969
48	18.11	10.39	-3.14	6.236
49	18.11	9.385	-3.14	6.545
50	18.11	8.385	-3.14	6.896
51	18.12	7.385	-3.14	7.285
52	18.12	6.385	-3.14	7.709
53	18.12	5.385	-3.14	8.162
54	18.12	4.385	-3.14	8.658
55	18.12	3.385	-3.14	9.178
56	18.12	2.385	-3.14	9.733
57	18.13	1.385	-3.14	10.3
58	18.13	1.385	-2.48	10.31
59	18.13	1.385	-1.82	10.32

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
60	18.13	1.385	-1.16	10.32
61	18.13	1.385	-0.5	10.33
62	18.13	1.385	0.16	10.33
63	18.13	1.385	0.82	10.32
64	18.13	1.385	1.48	10.31
65	17.3	1.46	1.48	9.984
66	16.47	1.536	1.48	9.684
67	15.64	1.611	1.48	9.412
68	14.81	1.687	1.48	9.197
69	13.98	1.763	1.48	9.02
70	13.15	1.838	1.48	8.891
71	12.32	1.914	1.48	8.813
72	11.49	1.989	1.48	8.806
73	10.66	2.065	1.48	8.856
74	9.827	2.14	1.48	8.957
75	8.997	2.216	1.48	9.112
76	8.167	2.291	1.48	9.313
77	7.337	2.367	1.48	9.562
78	6.507	2.443	1.48	9.86
79	5.677	2.518	1.48	10.21
80	4.847	2.594	1.48	10.61
81	4.017	2.669	1.48	11.02
82	3.188	2.745	1.48	11.49
83	2.358	2.82	1.48	12
84	1.528	2.896	1.48	12.54
Error cuadrático medio				5.1755

Tabla A.13: Error de pose en *MappingTB_v9_1*(0.5,0)

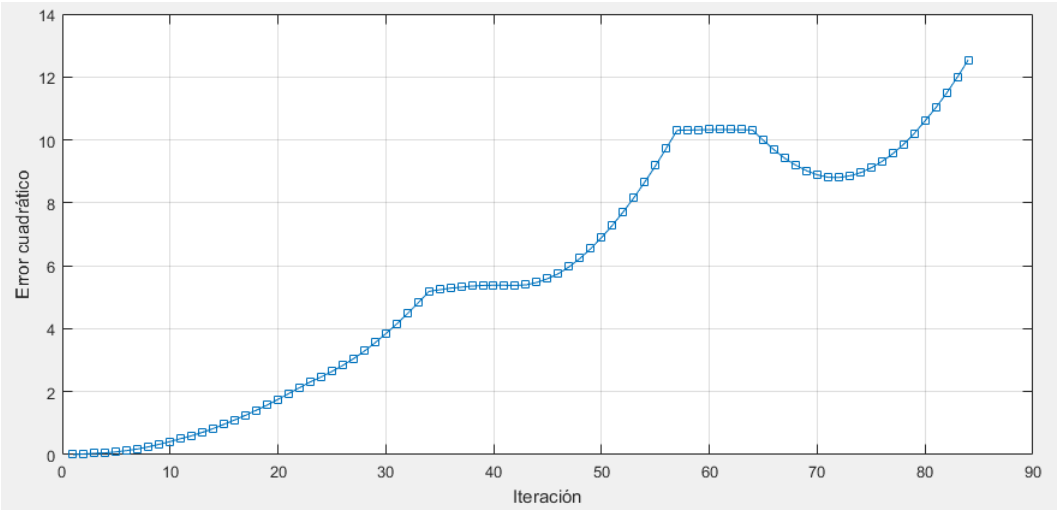


Figura A.71: Gráfica del error de pose en *MappingTB_v9_1*(0.5,0)

```
1 MappingTB_v9_1(0.8,0)
```

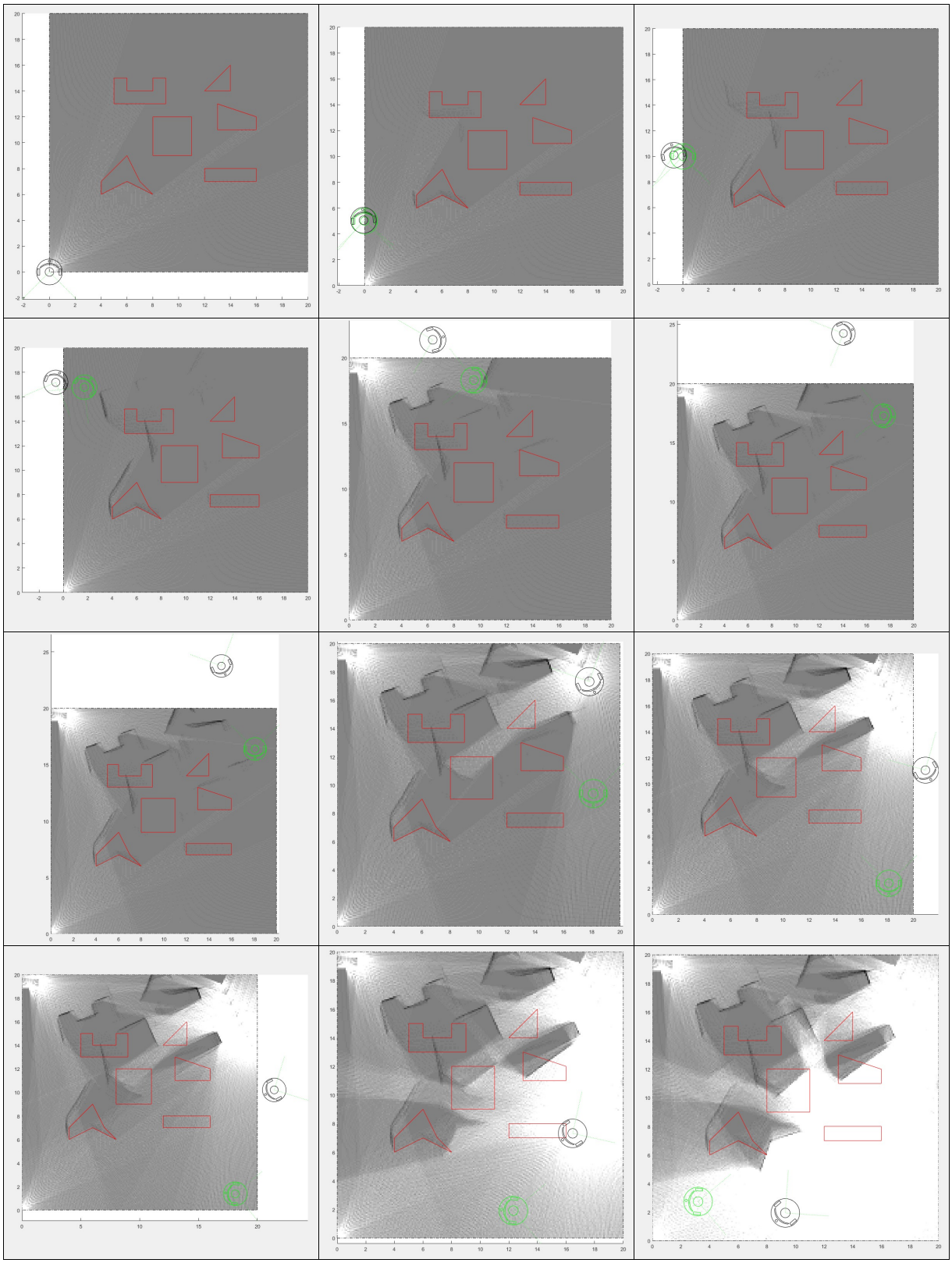


Figura A.72: Resultado de *MappingTB_v9_1* con error de odometría 0.8. Obstáculos en rojo, mapa en escala de grises, pose real en verde y pose calculada en negro.

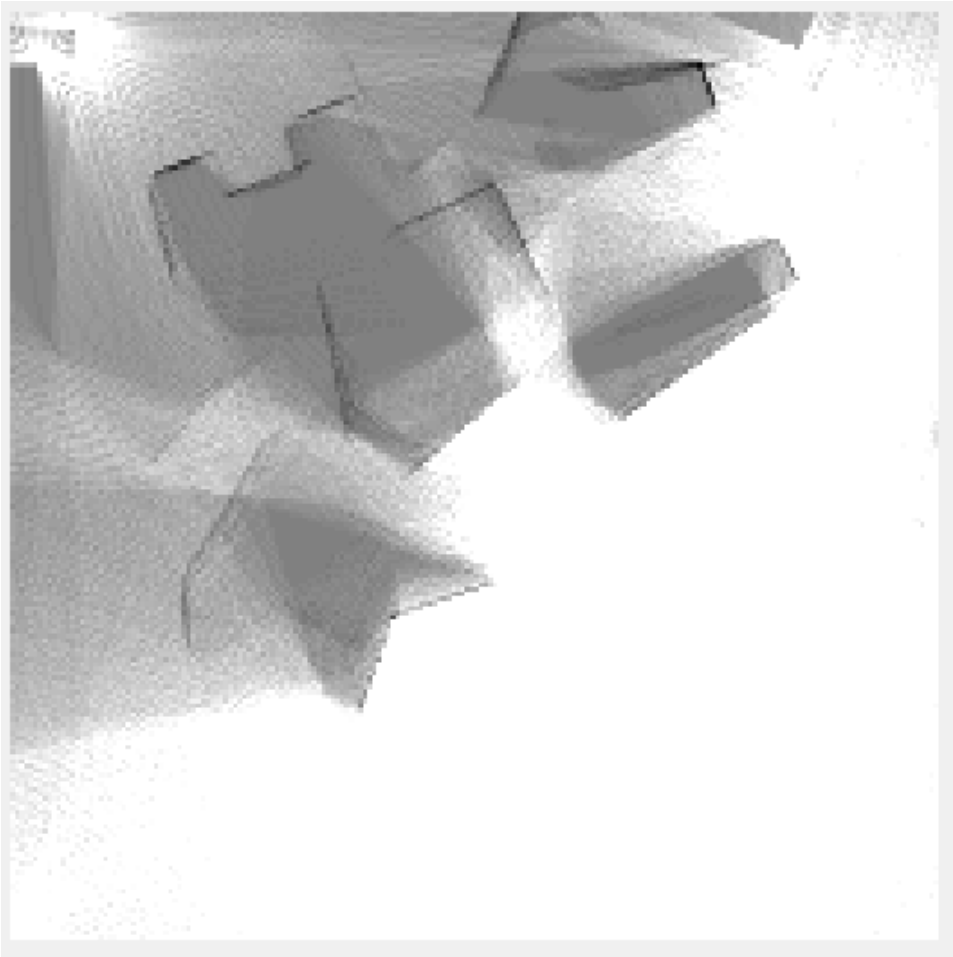


Figura A.73: Mapa resultante de *MappingTB_v1_2(0.8,0)*

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0	1	0	0.0245
3	0	2	0	0.0385
4	0	3	0	0.0549
5	0	4	0	0.0797
6	0	5	0	0.1374
7	0	6	0	0.2102
8	0	7	0	0.2994
9	0	8	0	0.4317
10	0	9	0	0.5862
11	0	10	0	0.7528
12	0	11	0	0.9455
13	0	12	0	1.161
14	0	13	0	1.396
15	0.179	14.04	-0.17	1.638
16	0.532	15.04	-0.34	1.879
17	1.049	15.96	-0.51	2.138

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
18	1.714	16.79	-0.68	2.392
19	2.509	17.49	-0.85	2.649
20	3.411	18.04	-1.02	2.915
21	4.394	18.43	-1.19	3.193
22	5.428	18.66	-1.36	3.423
23	6.486	18.7	-1.53	3.666
24	7.535	18.56	-1.7	3.899
25	8.527	18.43	-1.7	4.147
26	9.519	18.3	-1.7	4.42
27	10.51	18.18	-1.7	4.742
28	11.5	18.05	-1.7	5.091
29	12.49	17.92	-1.7	5.454
30	13.48	17.79	-1.7	5.85
31	14.48	17.66	-1.7	6.287
32	15.47	17.53	-1.7	6.744
33	16.46	17.4	-1.7	7.24
34	17.45	17.27	-1.7	7.766
35	17.59	17.23	-1.88	7.832
36	17.73	17.16	-2.06	7.884
37	17.84	17.06	-2.24	7.926
38	17.94	16.95	-2.42	7.952
39	18.02	16.82	-2.6	7.96
40	18.07	16.68	-2.78	7.965
41	18.1	16.54	-2.96	7.972
42	18.1	16.39	-3.14	7.961
43	18.1	15.39	-3.14	7.902
44	18.11	14.39	-3.14	7.861
45	18.11	13.39	-3.14	7.833
46	18.11	12.39	-3.14	7.836
47	18.11	11.39	-3.14	7.843
48	18.11	10.39	-3.14	7.873
49	18.11	9.385	-3.14	7.938
50	18.11	8.385	-3.14	8.015
51	18.12	7.385	-3.14	8.137
52	18.12	6.385	-3.14	8.279
53	18.12	5.385	-3.14	8.451
54	18.12	4.385	-3.14	8.649
55	18.12	3.385	-3.14	8.87
56	18.12	2.385	-3.14	9.133
57	18.13	1.385	-3.14	9.409
58	18.13	1.385	-2.48	9.407
59	18.13	1.385	-1.82	9.41
60	18.13	1.385	-1.16	9.421
61	18.13	1.385	-0.5	9.45
62	18.13	1.385	0.16	9.455

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
63	18.13	1.385	0.82	9.454
64	18.13	1.385	1.48	9.44
65	17.3	1.46	1.48	9.028
66	16.47	1.536	1.48	8.628
67	15.64	1.611	1.48	8.228
68	14.81	1.687	1.48	7.874
69	13.98	1.763	1.48	7.515
70	13.15	1.838	1.48	7.165
71	12.32	1.914	1.48	6.842
72	11.49	1.989	1.48	6.542
73	10.66	2.065	1.48	6.273
74	9.827	2.14	1.48	6.023
75	8.997	2.216	1.48	5.833
76	8.167	2.291	1.48	5.698
77	7.337	2.367	1.48	5.604
78	6.507	2.443	1.48	5.578
79	5.677	2.518	1.48	5.64
80	4.847	2.594	1.48	5.767
81	4.017	2.669	1.48	5.965
82	3.188	2.745	1.48	6.204
83	2.358	2.82	1.48	6.503
84	1.528	2.896	1.48	6.866
Error cuadrático medio				5.7017

Tabla A.14: Error de pose en *MappingTB_v9_1*(0.8,0)

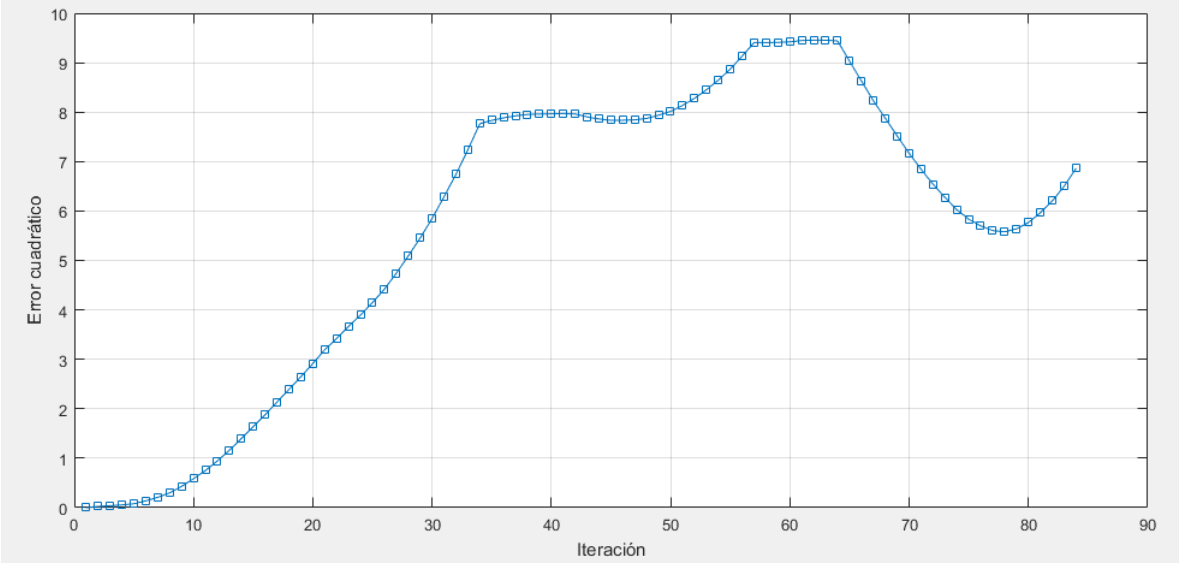


Figura A.74: Gráfica del error de pose en *MappingTB_v9_1*(0.8,0)

Profile Summary

Generated 05-Sep-2016 00:27:36 using performance time.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
MappingTB_v9_1	1	189.689 s	8.560 s	
Mapping_v8	84	142.374 s	10.773 s	
CalculateMAP_v3	2325	131.601 s	82.220 s	
VissionSimulation_v1	84	35.909 s	0.734 s	
mapshow	84	1.123 s	0.019 s	
dibujar_v2	168	0.572 s	0.483 s	

Figura A.75: Desempeño temporal de *MappingTB_v9_1(0.8,0)* con precisión 1

A.17. MappingTB_v9_2

```
1 MappingTB_v9_2(0.16,0)
```

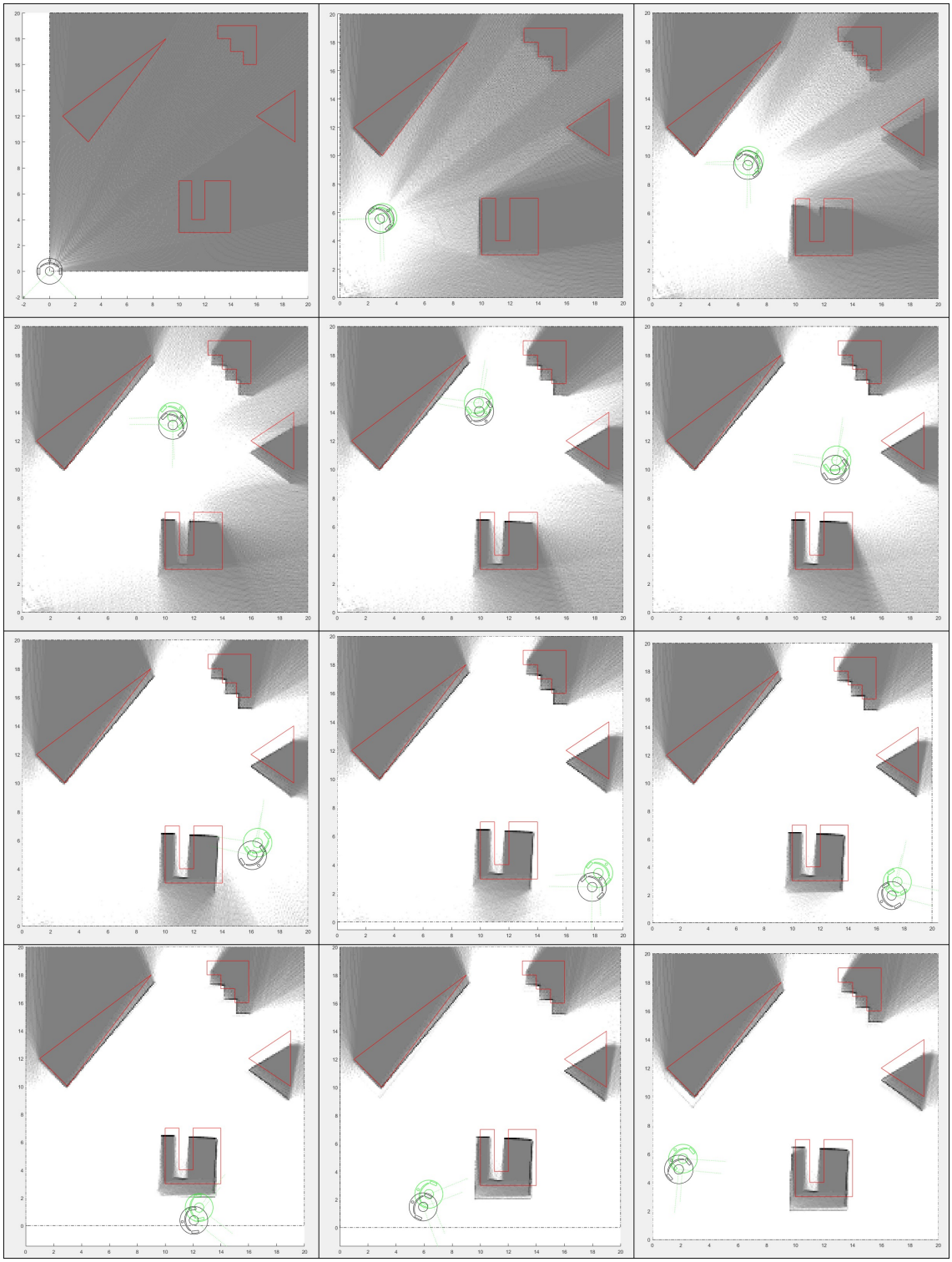


Figura A.76: Resultado de *MappingTB_v9_2* con error de odometría 0.16. Obstáculos en rojo, mapa en escala de grises, pose real en verde y pose calculada en negro.

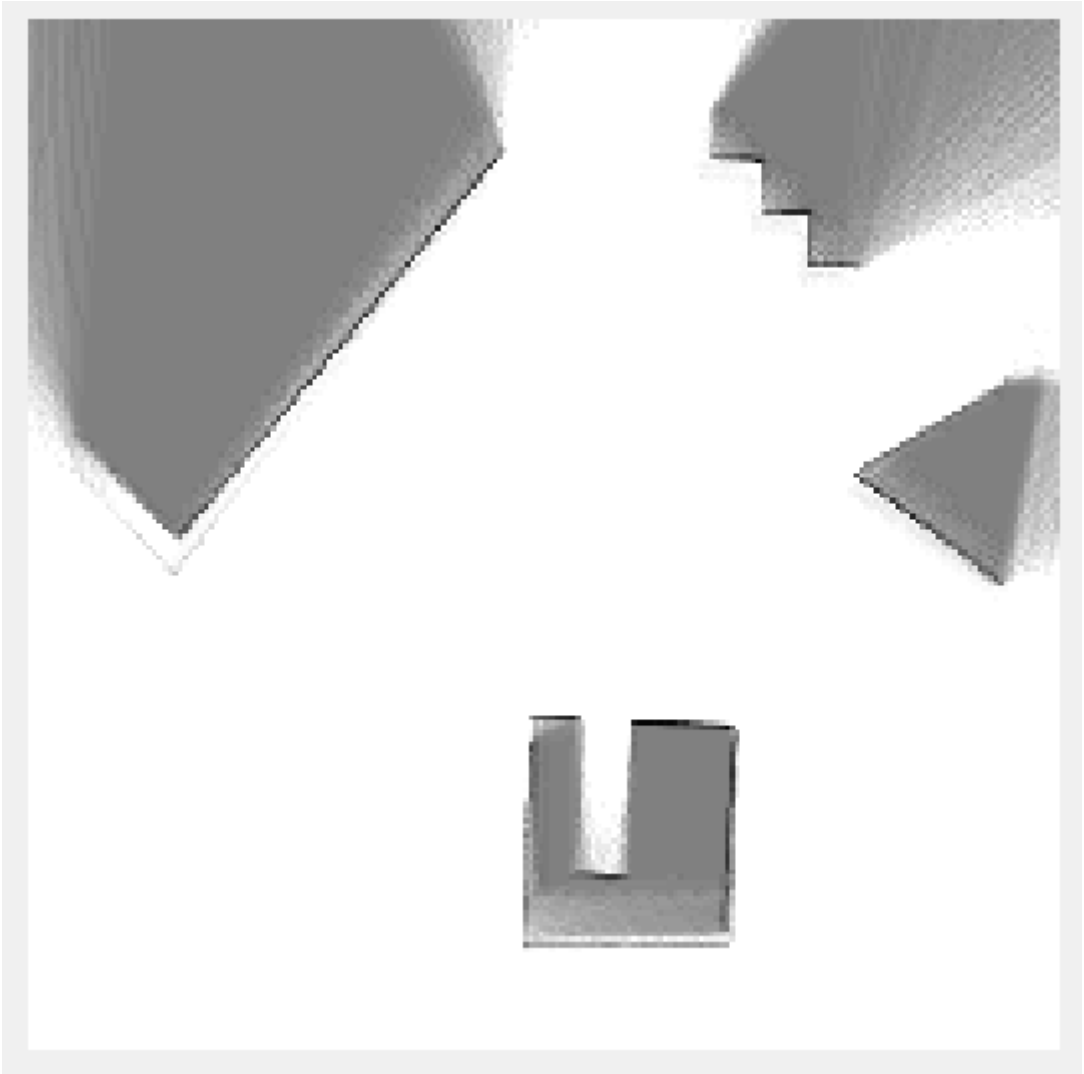


Figura A.77: Mapa resultante de *MappingTB_v9_2(0.16,0)*

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0.168	1.112	-0.15	0.0226
3	0.501	2.187	-0.3	0.0603
4	0.99	3.2	-0.45	0.1053
5	1.625	4.129	-0.6	0.1426
6	2.392	4.952	-0.75	0.1726
7	3.017	5.622	-0.75	0.2172
8	3.642	6.293	-0.75	0.2366
9	4.266	6.964	-0.75	0.2535
10	4.891	7.635	-0.75	0.2678
11	5.516	8.305	-0.75	0.2832
12	6.141	8.976	-0.75	0.3004
13	6.766	9.647	-0.75	0.3248
14	7.391	10.32	-0.75	0.3656

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
15	8.016	10.99	-0.75	0.395
16	8.64	11.66	-0.75	0.4526
17	9.265	12.33	-0.75	0.4985
18	9.89	13	-0.75	0.5278
19	10.52	13.67	-0.75	0.5673
20	11.14	14.34	-0.75	0.6291
21	10.88	14.21	-1.1	0.6152
22	10.59	14.17	-1.45	0.5953
23	10.31	14.24	-1.8	0.5768
24	10.06	14.4	-2.15	0.5583
25	9.888	14.63	-2.5	0.5734
26	10.49	13.83	-2.5	0.5787
27	11.08	13.03	-2.5	0.5948
28	11.68	12.23	-2.5	0.6151
29	12.28	11.43	-2.5	0.6436
30	12.88	10.63	-2.5	0.6585
31	13.48	9.827	-2.5	0.6844
32	14.08	9.026	-2.5	0.7225
33	14.68	8.225	-2.5	0.7808
34	15.27	7.424	-2.5	0.8287
35	15.87	6.622	-2.5	0.8872
36	16.47	5.821	-2.5	0.9605
37	17.07	5.02	-2.5	1.012
38	17.67	4.219	-2.5	1.057
39	18.27	3.418	-2.5	1.093
40	18.27	3.418	-1.91	1.092
41	18.27	3.418	-1.32	1.081
42	18.27	3.418	-0.73	1.076
43	18.27	3.418	-0.14	1.082
44	18.27	3.418	0.45	1.092
45	18.27	3.418	1.04	1.093
46	18.27	3.418	1.63	1.084
47	18.27	3.418	2.22	1.095
48	17.51	2.931	2.145	1.062
49	16.73	2.502	2.07	1.041
50	15.91	2.134	1.995	1.024
51	15.07	1.827	1.92	1.022
52	14.21	1.585	1.845	1.022
53	13.33	1.407	1.77	0.993
54	12.44	1.296	1.695	0.992
55	11.55	1.252	1.62	1.016
56	10.65	1.275	1.545	1.029
57	9.758	1.366	1.47	1.05
58	8.876	1.522	1.395	1.059
59	8.008	1.745	1.32	1.061

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
60	7.16	2.031	1.245	1.023
61	6.335	2.381	1.17	0.9904
62	5.539	2.791	1.095	0.963
63	4.775	3.26	1.02	0.9351
64	4.049	3.785	0.945	0.8931
65	3.364	4.362	0.87	0.8616
66	2.725	4.99	0.795	0.8382
67	2.134	5.663	0.72	0.8079
Error cuadrático medio				0.7199

Tabla A.15: Error de pose en *MappingTB_v9_2*(0.16,0)

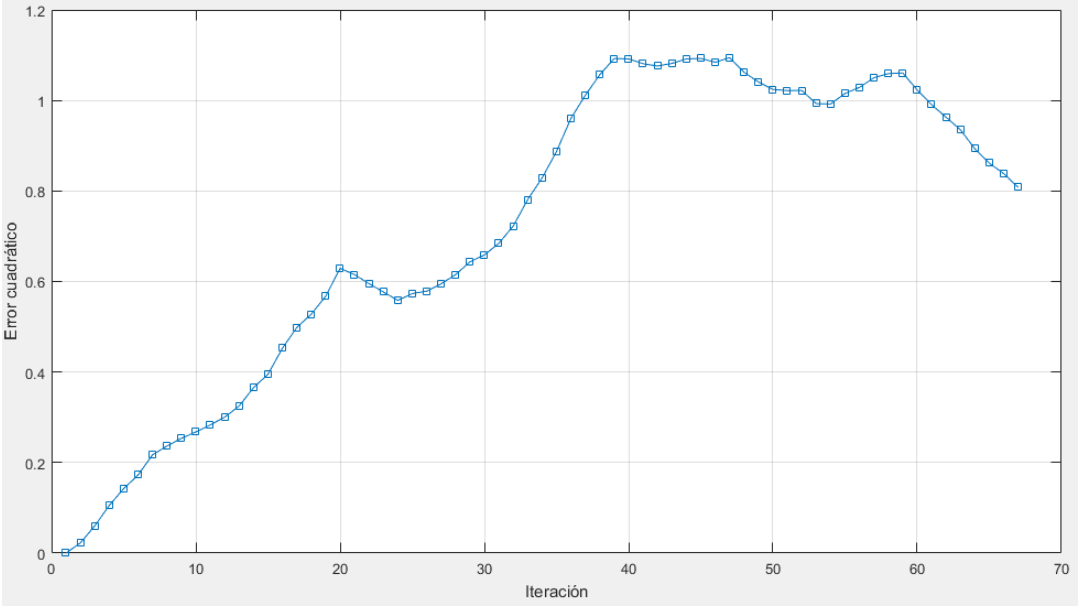


Figura A.78: Gráfica del error de pose en *MappingTB_v9_2*(0.16,0)

```
MappingTB_v9_2(0.3,0)
```

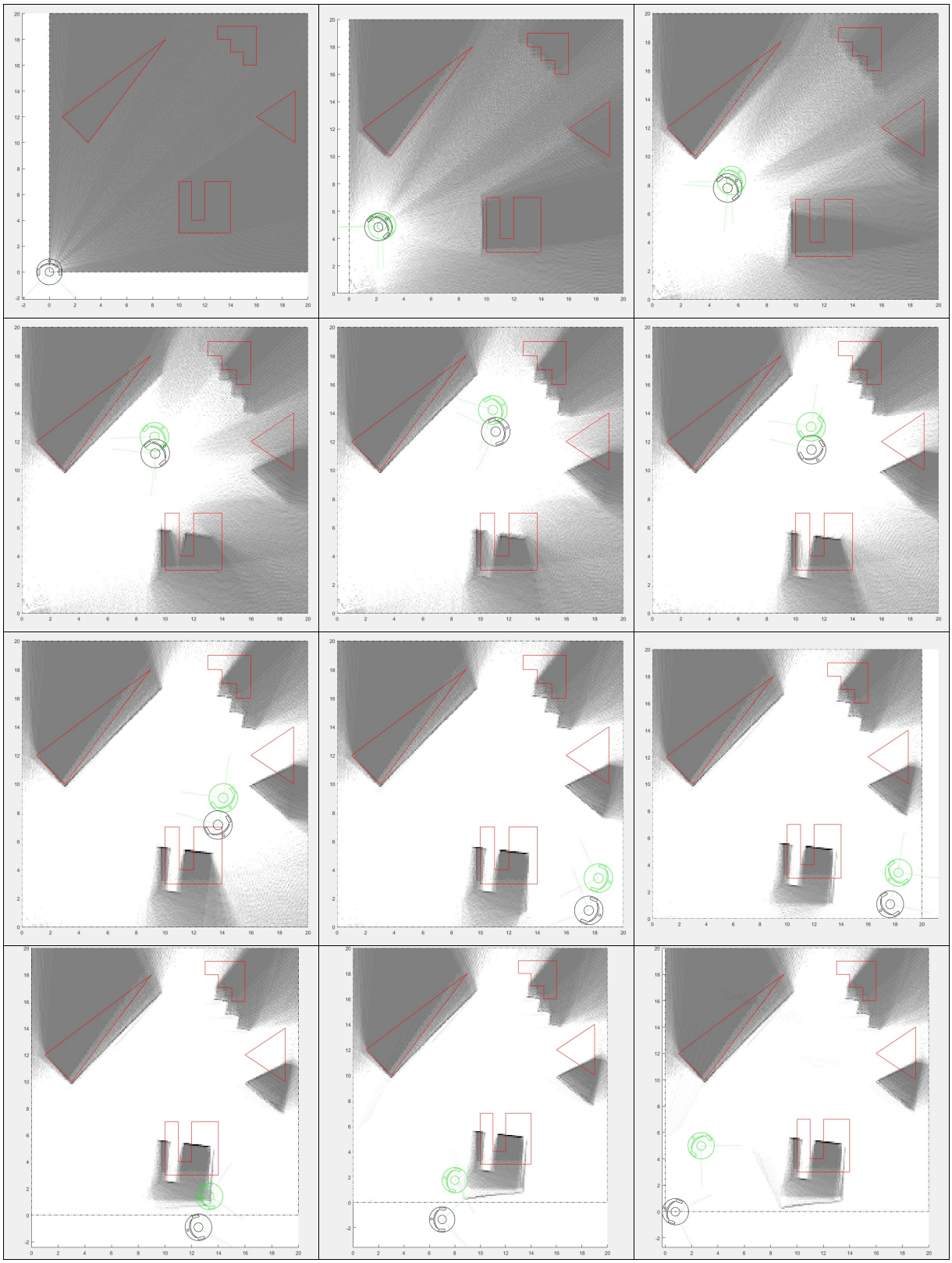


Figura A.79: Resultado de *MappingTB_v9_2* con error de odometría 0.3. Obstáculos en rojo, mapa en escala de grises, pose real en verde y pose calculada en negro.

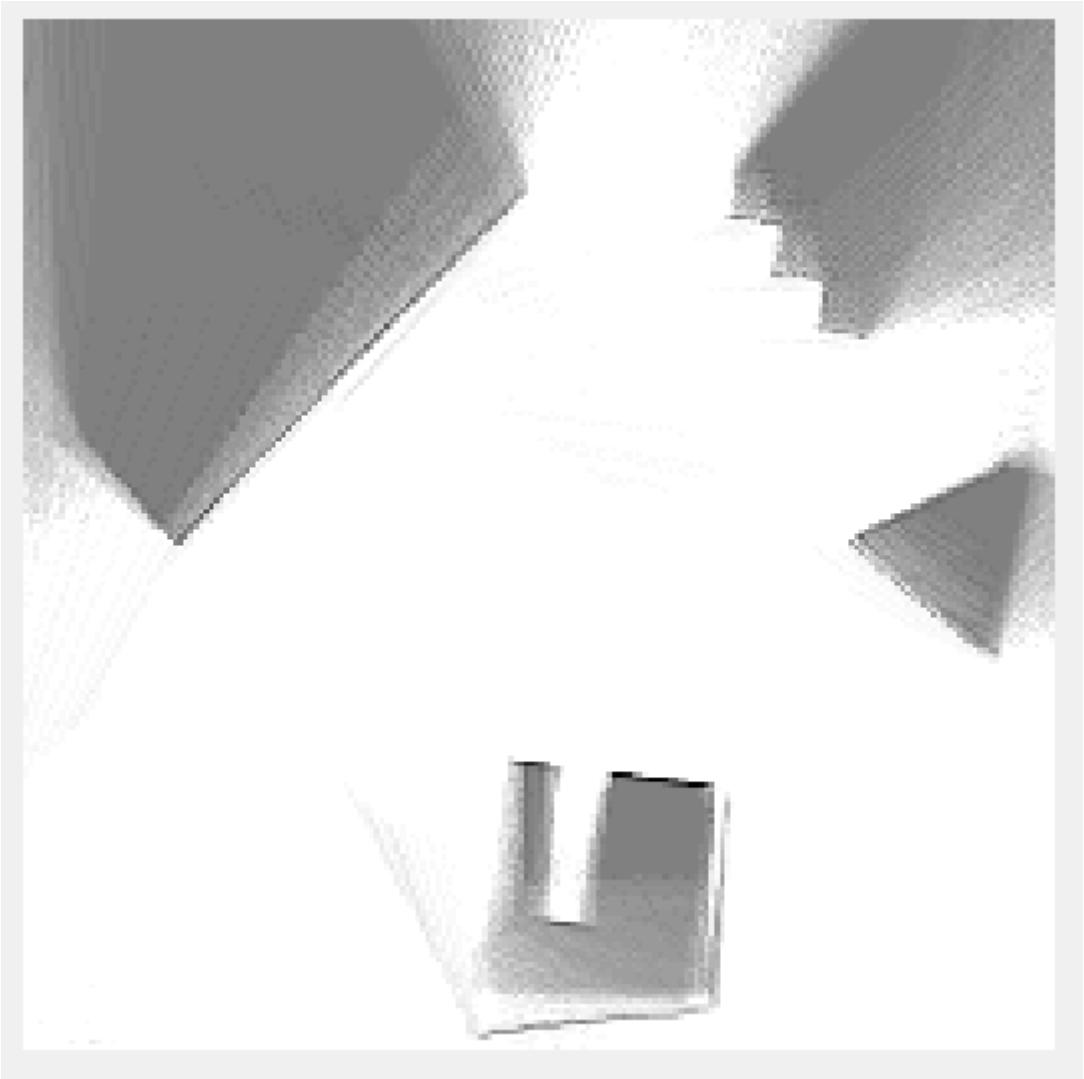


Figura A.80: Mapa resultante de *MappingTB_v9_2(0.3,0)*

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0.168	1.112	-0.15	0.0801
3	0.501	2.187	-0.3	0.1769
4	0.99	3.2	-0.45	0.2577
5	1.625	4.129	-0.6	0.2841
6	2.392	4.952	-0.75	0.3001
7	3.017	5.622	-0.75	0.3523
8	3.642	6.293	-0.75	0.3773
9	4.266	6.964	-0.75	0.4523
10	4.891	7.635	-0.75	0.4917
11	5.516	8.305	-0.75	0.5911
12	6.141	8.976	-0.75	0.6677
13	6.766	9.647	-0.75	0.7353
14	7.391	10.32	-0.75	0.8514

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
15	8.016	10.99	-0.75	0.963
16	8.64	11.66	-0.75	1.052
17	9.265	12.33	-0.75	1.173
18	9.89	13	-0.75	1.311
19	10.52	13.67	-0.75	1.459
20	11.14	14.34	-0.75	1.576
21	10.88	14.21	-1.1	1.533
22	10.59	14.17	-1.45	1.498
23	10.31	14.24	-1.8	1.498
24	10.06	14.4	-2.15	1.454
25	9.888	14.63	-2.5	1.438
26	10.49	13.83	-2.5	1.532
27	11.08	13.03	-2.5	1.601
28	11.68	12.23	-2.5	1.651
29	12.28	11.43	-2.5	1.682
30	12.88	10.63	-2.5	1.758
31	13.48	9.827	-2.5	1.841
32	14.08	9.026	-2.5	1.932
33	14.68	8.225	-2.5	2.008
34	15.27	7.424	-2.5	2.074
35	15.87	6.622	-2.5	2.134
36	16.47	5.821	-2.5	2.201
37	17.07	5.02	-2.5	2.258
38	17.67	4.219	-2.5	2.283
39	18.27	3.418	-2.5	2.35
40	18.27	3.418	-1.91	2.357
41	18.27	3.418	-1.32	2.358
42	18.27	3.418	-0.73	2.368
43	18.27	3.418	-0.14	2.39
44	18.27	3.418	0.45	2.422
45	18.27	3.418	1.04	2.432
46	18.27	3.418	1.63	2.427
47	18.27	3.418	2.22	2.429
48	17.51	2.931	2.145	2.429
49	16.73	2.502	2.07	2.426
50	15.91	2.134	1.995	2.385
51	15.07	1.827	1.92	2.371
52	14.21	1.585	1.845	2.387
53	13.33	1.407	1.77	2.459
54	12.44	1.296	1.695	2.525
55	11.55	1.252	1.62	2.63
56	10.65	1.275	1.545	2.767
57	9.758	1.366	1.47	2.899
58	8.876	1.522	1.395	3.061
59	8.008	1.745	1.32	3.255

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
60	7.16	2.031	1.245	3.491
61	6.335	2.381	1.17	3.764
62	5.539	2.791	1.095	4.061
63	4.775	3.26	1.02	4.354
64	4.049	3.785	0.945	4.668
65	3.364	4.362	0.87	5.025
66	2.725	4.99	0.795	5.389
67	2.134	5.663	0.72	5.8
Error cuadrático medio				2.018

Tabla A.16: Error de pose en *MappingTB_v9_2*(0.3,0)

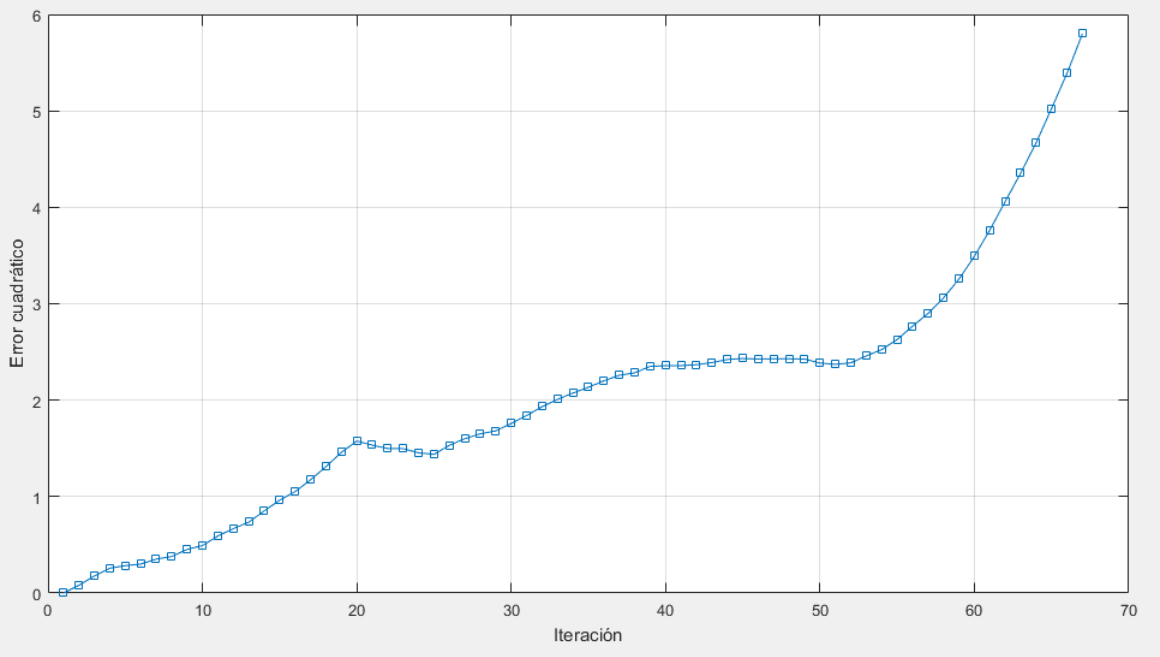


Figura A.81: Gráfica del error de pose en *MappingTB_v9_2*(0.3,0)

A.18. MappingTB_v10_1

```
1 MappingTB_v10_1(0.5,0)
```

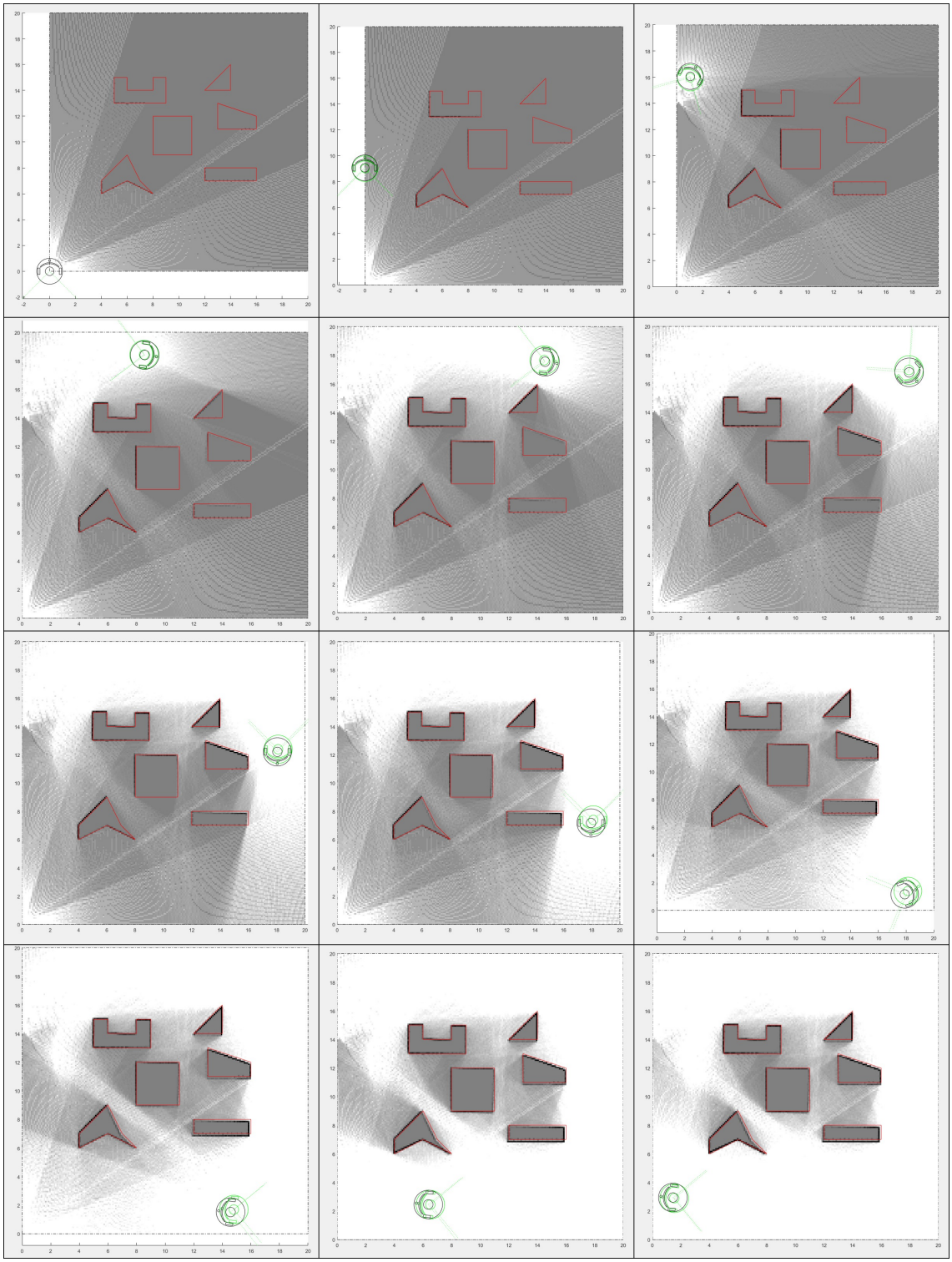


Figura A.82: Resultado de *MappingTB_v10_1* con error de odometría 0.5. Obstáculos en rojo, mapa en escala de grises, pose real en verde y pose calculada en negro.

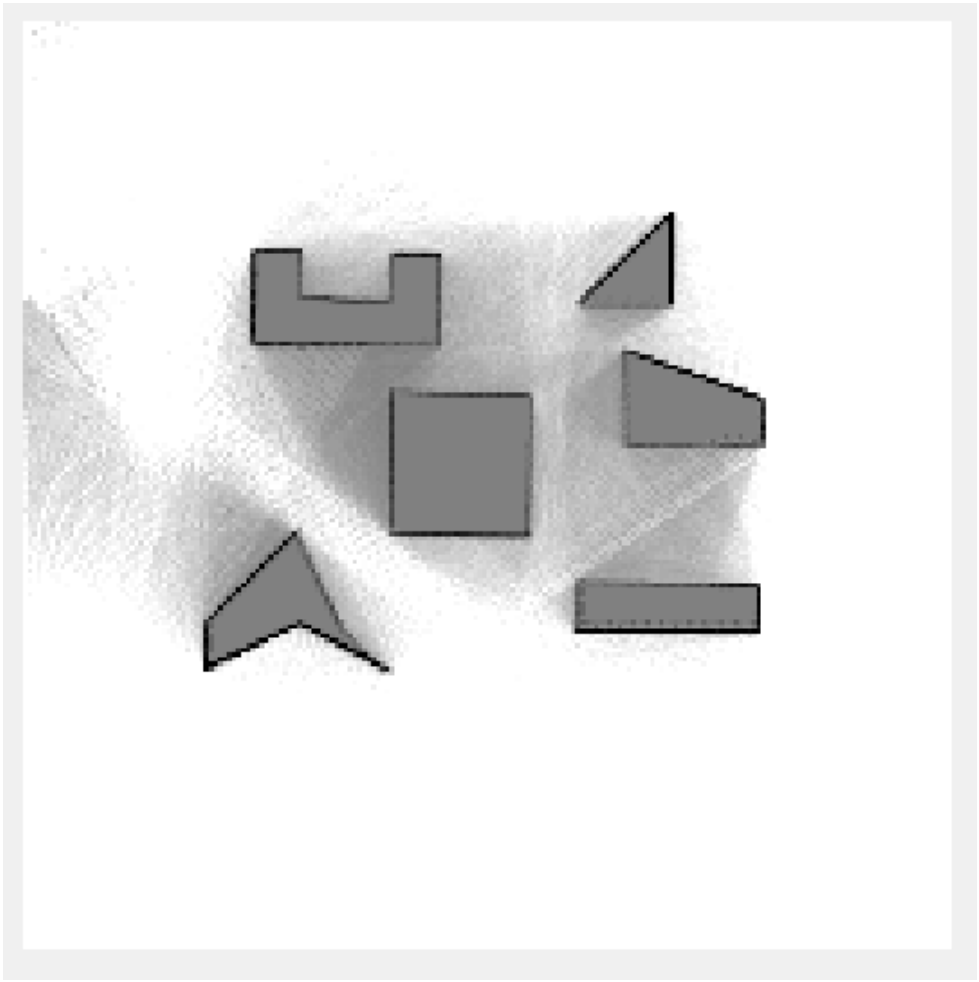


Figura A.83: Mapa resultante de *MappingTB_v10_1(0.5,0)*

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0	1	0	0.0147
3	0	2	0	0.0135
4	0	3	0	0.0342
5	0	4	0	0.045
6	0	5	0	0.0505
7	0	6	0	0.05
8	0	7	0	0.05
9	0	8	0	0.0546
10	0	9	0	0.067
11	0	10	0	0.0792
12	0	11	0	0.0768
13	0	12	0	0.0743
14	0	13	0	0.0783
15	0.179	14.04	-0.17	0.0768
16	0.532	15.04	-0.34	0.103

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
17	1.049	15.96	-0.51	0.1018
18	1.714	16.79	-0.68	0.1046
19	2.509	17.49	-0.85	0.0741
20	3.411	18.04	-1.02	0.0582
21	4.394	18.43	-1.19	0.0523
22	5.428	18.66	-1.36	0.0619
23	6.486	18.7	-1.53	0.0627
24	7.535	18.56	-1.7	0.0554
25	8.527	18.43	-1.7	0.056
26	9.519	18.3	-1.7	0.0587
27	10.51	18.18	-1.7	0.0739
28	11.5	18.05	-1.7	0.0728
29	12.49	17.92	-1.7	0.0851
30	13.48	17.79	-1.7	0.1101
31	14.48	17.66	-1.7	0.1249
32	15.47	17.53	-1.7	0.1304
33	16.46	17.4	-1.7	0.1445
34	17.45	17.27	-1.7	0.1531
35	17.59	17.23	-1.88	0.1593
36	17.73	17.16	-2.06	0.1616
37	17.84	17.06	-2.24	0.1686
38	17.94	16.95	-2.42	0.1664
39	18.02	16.82	-2.6	0.1723
40	18.07	16.68	-2.78	0.1705
41	18.1	16.54	-2.96	0.1784
42	18.1	16.39	-3.14	0.1723
43	18.1	15.39	-3.14	0.1844
44	18.11	14.39	-3.14	0.1793
45	18.11	13.39	-3.14	0.185
46	18.11	12.39	-3.14	0.1889
47	18.11	11.39	-3.14	0.21
48	18.11	10.39	-3.14	0.2367
49	18.11	9.385	-3.14	0.2403
50	18.11	8.385	-3.14	0.2604
51	18.12	7.385	-3.14	0.2672
52	18.12	6.385	-3.14	0.2652
53	18.12	5.385	-3.14	0.2831
54	18.12	4.385	-3.14	0.3101
55	18.12	3.385	-3.14	0.3149
56	18.12	2.385	-3.14	0.3326
57	18.13	1.385	-3.14	0.3366
58	18.13	1.385	-2.48	0.3382
59	18.13	1.385	-1.82	0.3382
60	18.13	1.385	-1.16	0.3276
61	18.13	1.385	-0.5	0.3185

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
62	18.13	1.385	0.16	0.3205
63	18.13	1.385	0.82	0.3206
64	18.13	1.385	1.48	0.3272
65	17.3	1.46	1.48	0.3111
66	16.47	1.536	1.48	0.3041
67	15.64	1.611	1.48	0.2889
68	14.81	1.687	1.48	0.2865
69	13.98	1.763	1.48	0.2808
70	13.15	1.838	1.48	0.2796
71	12.32	1.914	1.48	0.2571
72	11.49	1.989	1.48	0.2388
73	10.66	2.065	1.48	0.2191
74	9.827	2.14	1.48	0.2003
75	8.997	2.216	1.48	0.1848
76	8.167	2.291	1.48	0.1709
77	7.337	2.367	1.48	0.1504
78	6.507	2.443	1.48	0.1491
79	5.677	2.518	1.48	0.1427
80	4.847	2.594	1.48	0.1388
81	4.017	2.669	1.48	0.1463
82	3.188	2.745	1.48	0.1275
83	2.358	2.82	1.48	0.1184
84	1.528	2.896	1.48	0.1127
Error cuadrático medio				0.1666

Tabla A.17: Error de pose en *MappingTB_v10_1*(0.5,0)

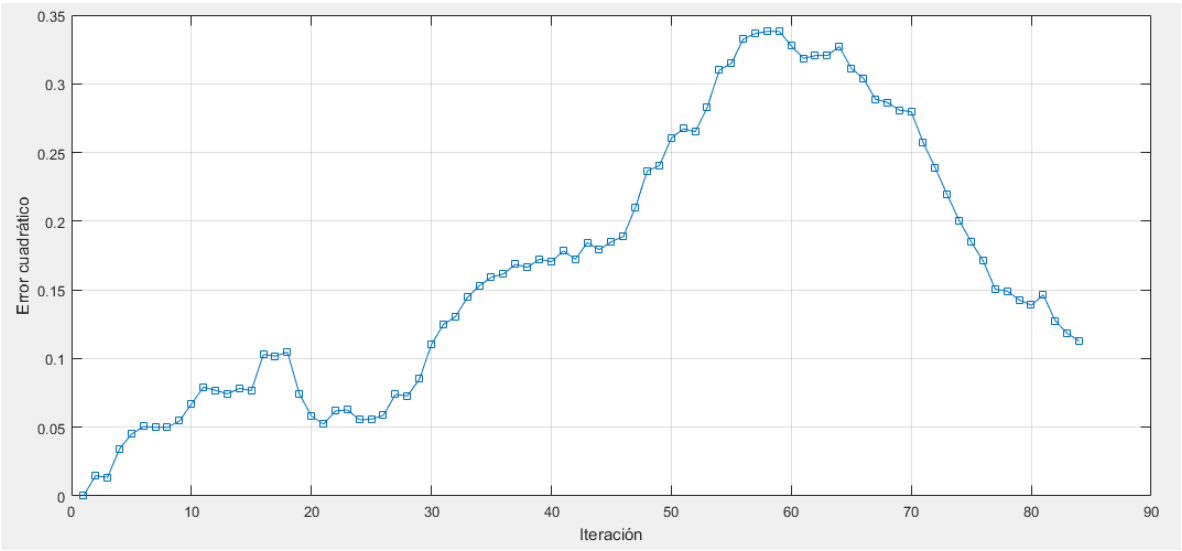


Figura A.84: Gráfica del error de pose en *MappingTB_v10_1*(0.5,0)

```
1 MappingTB_v10_1(0.5,0.1)
```

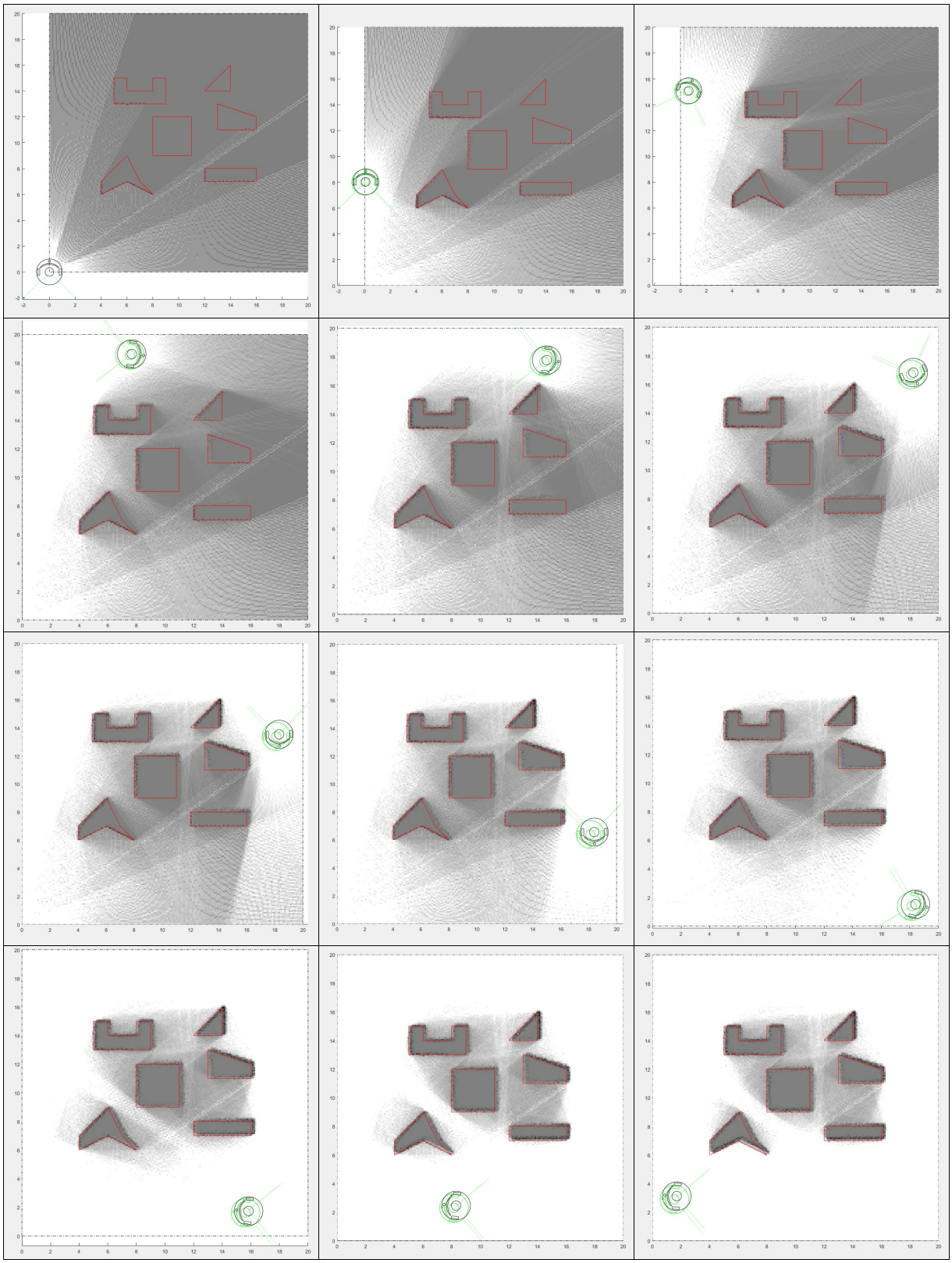


Figura A.85: Resultado de *MappingTB_v10_1* con errores de odometría 0.5 y de distancia 0.1. Obstáculos en rojo, mapa en escala de grises y pose en negro.

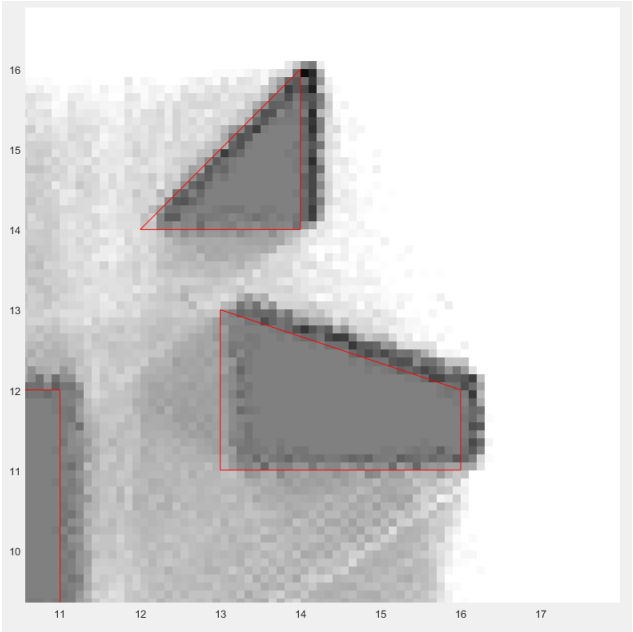


Figura A.86: Ampliación del resultado final de *MappingTB_v10_1* con errores de odometría 0.5 y de distancia 0.1

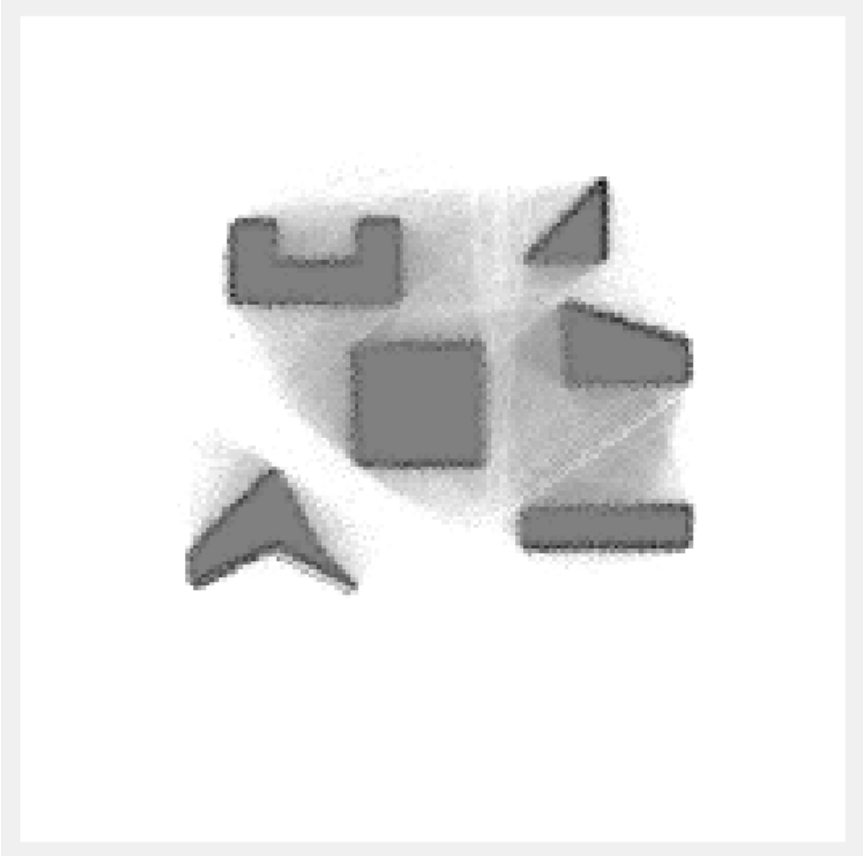


Figura A.87: Mapa resultante de *MappingTB_v10_1(0.5,0.1)*

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0	1	0	0.0135
3	0	2	0	0.0328
4	0	3	0	0.0444
5	0	4	0	0.0642
6	0	5	0	0.0805
7	0	6	0	0.0912
8	0	7	0	0.0966
9	0	8	0	0.0913
10	0	9	0	0.0834
11	0	10	0	0.0884
12	0	11	0	0.077
13	0	12	0	0.0794
14	0	13	0	0.0958
15	0.179	14.04	-0.17	0.1096
16	0.532	15.04	-0.34	0.1204
17	1.049	15.96	-0.51	0.1339
18	1.714	16.79	-0.68	0.1233
19	2.509	17.49	-0.85	0.1326
20	3.411	18.04	-1.02	0.1425
21	4.394	18.43	-1.19	0.1362
22	5.428	18.66	-1.36	0.1327
23	6.486	18.7	-1.53	0.1394
24	7.535	18.56	-1.7	0.1505
25	8.527	18.43	-1.7	0.1633
26	9.519	18.3	-1.7	0.1678
27	10.51	18.18	-1.7	0.1746
28	11.5	18.05	-1.7	0.1658
29	12.49	17.92	-1.7	0.1954
30	13.48	17.79	-1.7	0.2077
31	14.48	17.66	-1.7	0.2073
32	15.47	17.53	-1.7	0.1965
33	16.46	17.4	-1.7	0.2175
34	17.45	17.27	-1.7	0.2221
35	17.59	17.23	-1.88	0.2244
36	17.73	17.16	-2.06	0.2349
37	17.84	17.06	-2.24	0.2259
38	17.94	16.95	-2.42	0.2201
39	18.02	16.82	-2.6	0.2183
40	18.07	16.68	-2.78	0.2174
41	18.1	16.54	-2.96	0.2095
42	18.1	16.39	-3.14	0.21
43	18.1	15.39	-3.14	0.2083
44	18.11	14.39	-3.14	0.2319
45	18.11	13.39	-3.14	0.2574

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
46	18.11	12.39	-3.14	0.2777
47	18.11	11.39	-3.14	0.2869
48	18.11	10.39	-3.14	0.2909
49	18.11	9.385	-3.14	0.306
50	18.11	8.385	-3.14	0.3112
51	18.12	7.385	-3.14	0.3148
52	18.12	6.385	-3.14	0.3341
53	18.12	5.385	-3.14	0.3253
54	18.12	4.385	-3.14	0.3287
55	18.12	3.385	-3.14	0.3173
56	18.12	2.385	-3.14	0.3079
57	18.13	1.385	-3.14	0.2917
58	18.13	1.385	-2.48	0.2932
59	18.13	1.385	-1.82	0.2938
60	18.13	1.385	-1.16	0.2864
61	18.13	1.385	-0.5	0.2929
62	18.13	1.385	0.16	0.2925
63	18.13	1.385	0.82	0.2955
64	18.13	1.385	1.48	0.2866
65	17.3	1.46	1.48	0.2665
66	16.47	1.536	1.48	0.2607
67	15.64	1.611	1.48	0.2449
68	14.81	1.687	1.48	0.2455
69	13.98	1.763	1.48	0.2429
70	13.15	1.838	1.48	0.2448
71	12.32	1.914	1.48	0.2418
72	11.49	1.989	1.48	0.2344
73	10.66	2.065	1.48	0.2307
74	9.827	2.14	1.48	0.2175
75	8.997	2.216	1.48	0.2164
76	8.167	2.291	1.48	0.2166
77	7.337	2.367	1.48	0.2131
78	6.507	2.443	1.48	0.2298
79	5.677	2.518	1.48	0.2374
80	4.847	2.594	1.48	0.2568
81	4.017	2.669	1.48	0.254
82	3.188	2.745	1.48	0.2535
83	2.358	2.82	1.48	0.2644
84	1.528	2.896	1.48	0.2806
Error cuadrático medio				0.205

Tabla A.18: Error de pose en *MappingTB_v10_1*(0.5,0.1)

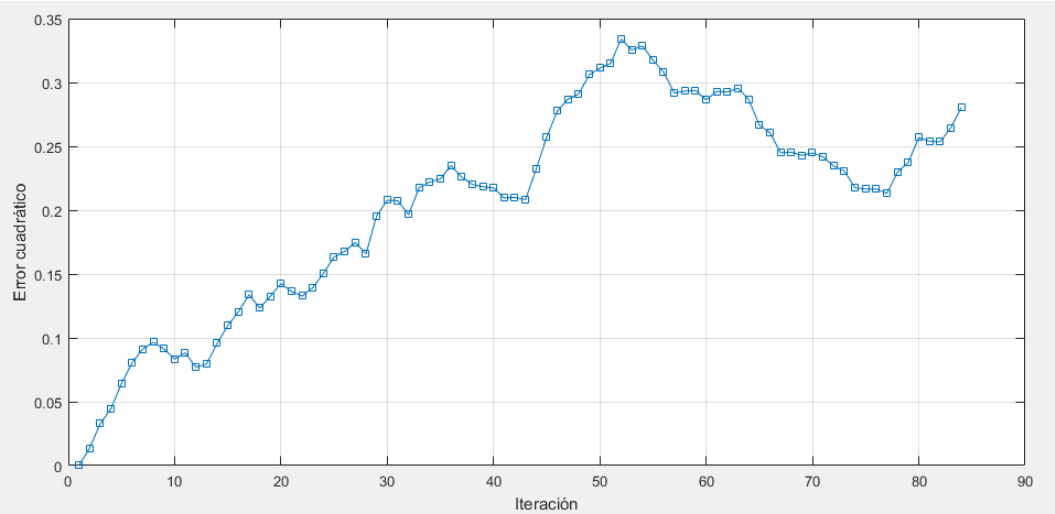


Figura A.88: Gráfica del error de pose en *MappingTB_v10_1*(0.5,0.1)

```
1 MappingTB_v10_1(0.8,0)
```

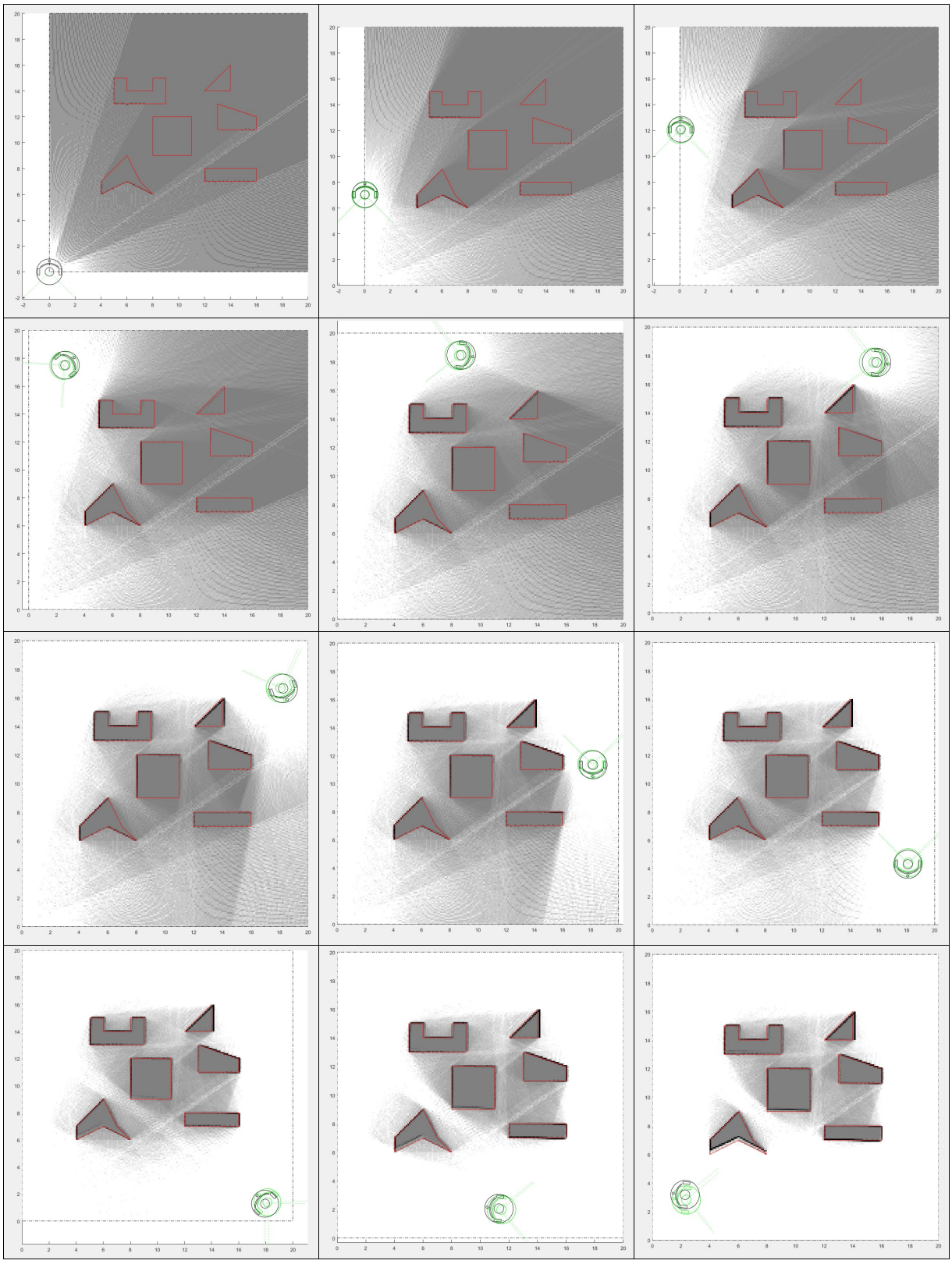


Figura A.89: Resultado de *MappingTB_v10_1* con error de odometría 0.8. Obstáculos en rojo, mapa en escala de grises, pose real en verde y pose calculada en negro.

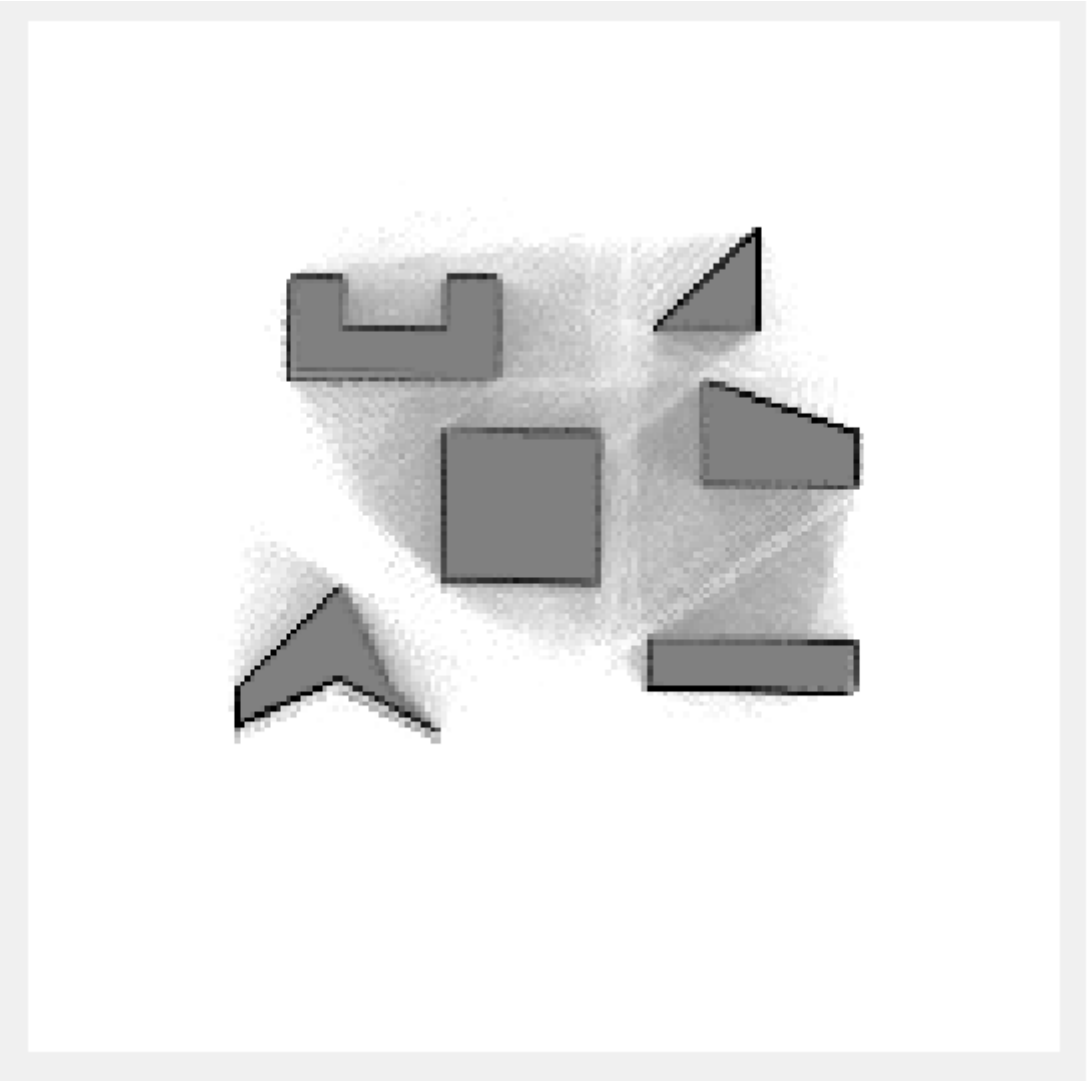


Figura A.90: Mapa resultante de *MappingTB_v10_1(0.8,0)*

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0	1	0	0.0278
3	0	2	0	0.0199
4	0	3	0	0.0412
5	0	4	0	0.03
6	0	5	0	0.0142
7	0	6	0	0.027
8	0	7	0	0.0483
9	0	8	0	0.0556
10	0	9	0	0.0594
11	0	10	0	0.0807
12	0	11	0	0.0975
13	0	12	0	0.125
14	0	13	0	0.117

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
15	0.179	14.04	-0.17	0.1014
16	0.532	15.04	-0.34	0.0952
17	1.049	15.96	-0.51	0.1291
18	1.714	16.79	-0.68	0.1055
19	2.509	17.49	-0.85	0.1166
20	3.411	18.04	-1.02	0.1201
21	4.394	18.43	-1.19	0.145
22	5.428	18.66	-1.36	0.156
23	6.486	18.7	-1.53	0.1718
24	7.535	18.56	-1.7	0.1673
25	8.527	18.43	-1.7	0.1448
26	9.519	18.3	-1.7	0.1396
27	10.51	18.18	-1.7	0.1234
28	11.5	18.05	-1.7	0.1501
29	12.49	17.92	-1.7	0.1715
30	13.48	17.79	-1.7	0.1717
31	14.48	17.66	-1.7	0.1876
32	15.47	17.53	-1.7	0.2001
33	16.46	17.4	-1.7	0.2024
34	17.45	17.27	-1.7	0.1961
35	17.59	17.23	-1.88	0.1983
36	17.73	17.16	-2.06	0.1919
37	17.84	17.06	-2.24	0.1926
38	17.94	16.95	-2.42	0.1988
39	18.02	16.82	-2.6	0.1901
40	18.07	16.68	-2.78	0.1907
41	18.1	16.54	-2.96	0.1925
42	18.1	16.39	-3.14	0.1883
43	18.1	15.39	-3.14	0.1649
44	18.11	14.39	-3.14	0.1256
45	18.11	13.39	-3.14	0.1151
46	18.11	12.39	-3.14	0.1042
47	18.11	11.39	-3.14	0.0982
48	18.11	10.39	-3.14	0.0976
49	18.11	9.385	-3.14	0.091
50	18.11	8.385	-3.14	0.1044
51	18.12	7.385	-3.14	0.1105
52	18.12	6.385	-3.14	0.1119
53	18.12	5.385	-3.14	0.0981
54	18.12	4.385	-3.14	0.1123
55	18.12	3.385	-3.14	0.133
56	18.12	2.385	-3.14	0.1596
57	18.13	1.385	-3.14	0.1912
58	18.13	1.385	-2.48	0.1922
59	18.13	1.385	-1.82	0.1905

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
60	18.13	1.385	-1.16	0.2067
61	18.13	1.385	-0.5	0.1864
62	18.13	1.385	0.16	0.189
63	18.13	1.385	0.82	0.2025
64	18.13	1.385	1.48	0.1975
65	17.3	1.46	1.48	0.1906
66	16.47	1.536	1.48	0.1597
67	15.64	1.611	1.48	0.1771
68	14.81	1.687	1.48	0.1813
69	13.98	1.763	1.48	0.1681
70	13.15	1.838	1.48	0.1892
71	12.32	1.914	1.48	0.1913
72	11.49	1.989	1.48	0.1931
73	10.66	2.065	1.48	0.2008
74	9.827	2.14	1.48	0.2003
75	8.997	2.216	1.48	0.2116
76	8.167	2.291	1.48	0.2119
77	7.337	2.367	1.48	0.238
78	6.507	2.443	1.48	0.2521
79	5.677	2.518	1.48	0.283
80	4.847	2.594	1.48	0.2948
81	4.017	2.669	1.48	0.32
82	3.188	2.745	1.48	0.3429
83	2.358	2.82	1.48	0.3726
84	1.528	2.896	1.48	0.4001
Error cuadrático medio				0.1577

Tabla A.19: Error de pose en *MappingTB_v10_1*(0.8,0)

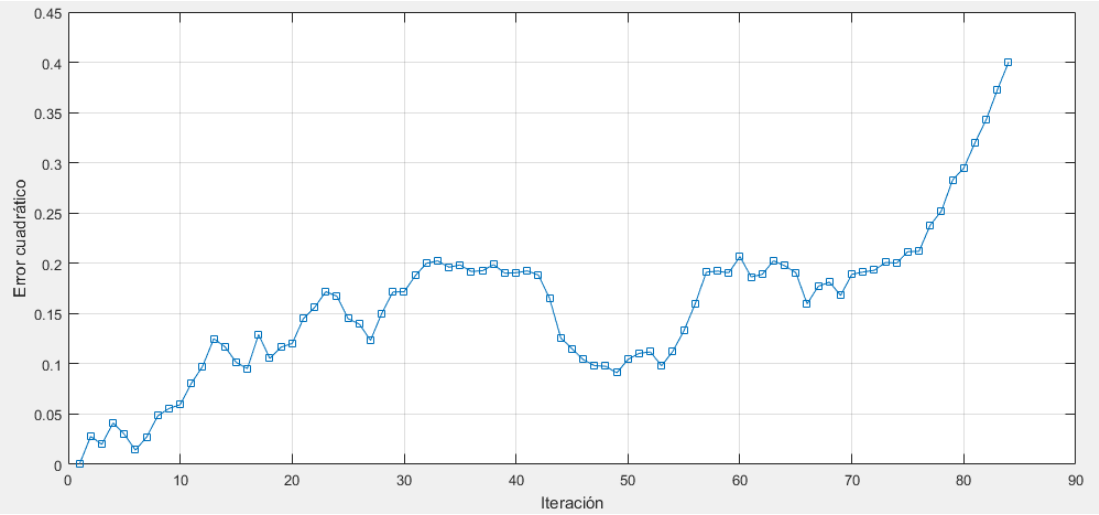


Figura A.91: Gráfica del error de pose en *MappingTB_v10_1*(0.8,0)

```
1 MappingTB_v10_1(0.8,0.1)
```

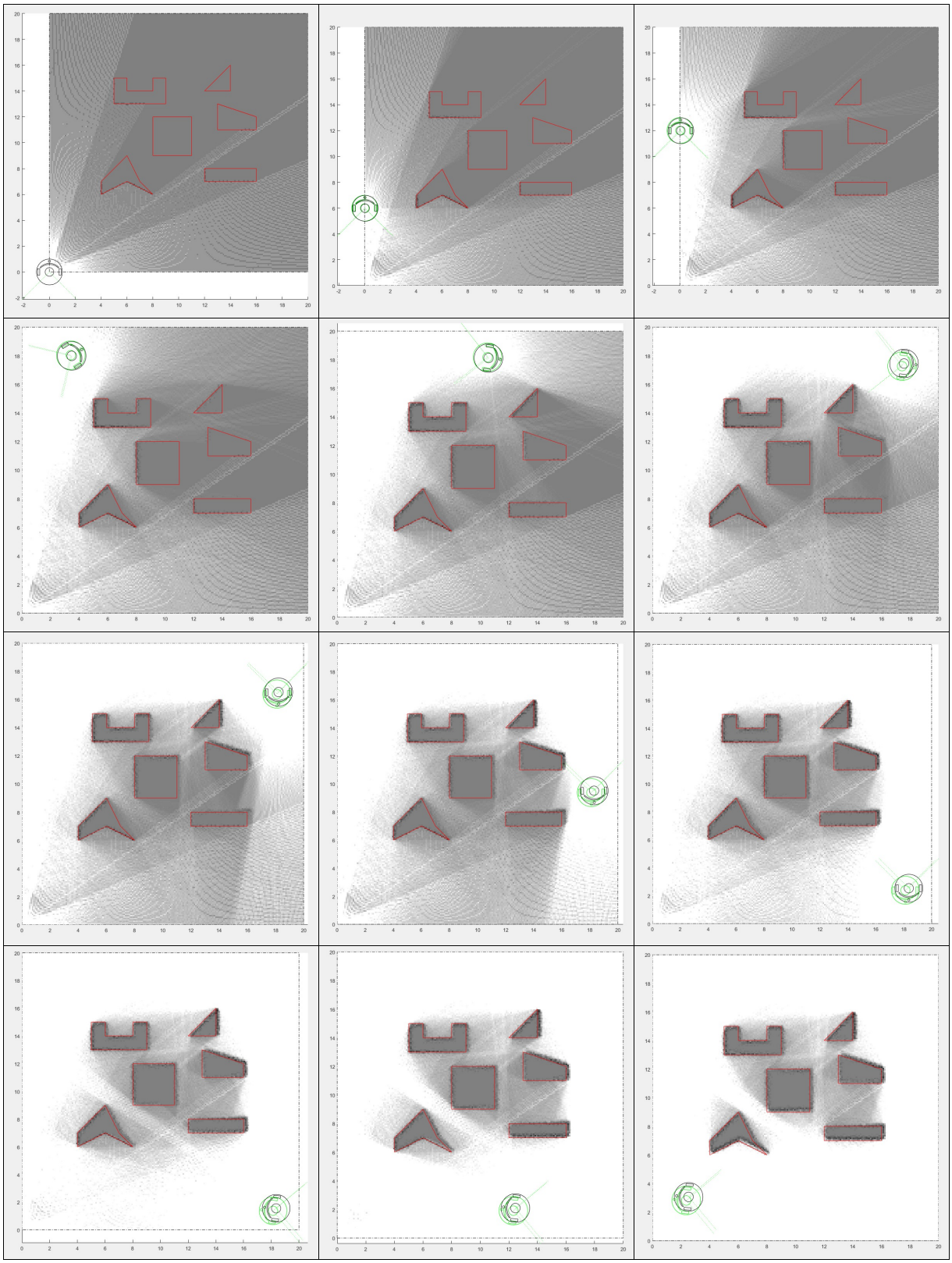


Figura A.92: Resultado de *MappingTB_v10_1* con errores de odometría 0.8 y de distancia 0.1. Obstáculos en rojo, mapa en escala de grises y pose en negro.

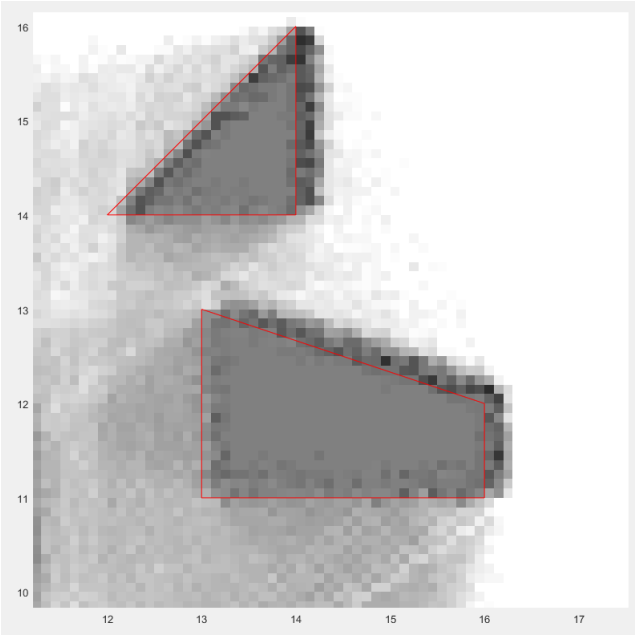


Figura A.93: Ampliación del resultado final de *MappingTB_v10_1* con errores de odometría 0.8 y de distancia 0.1

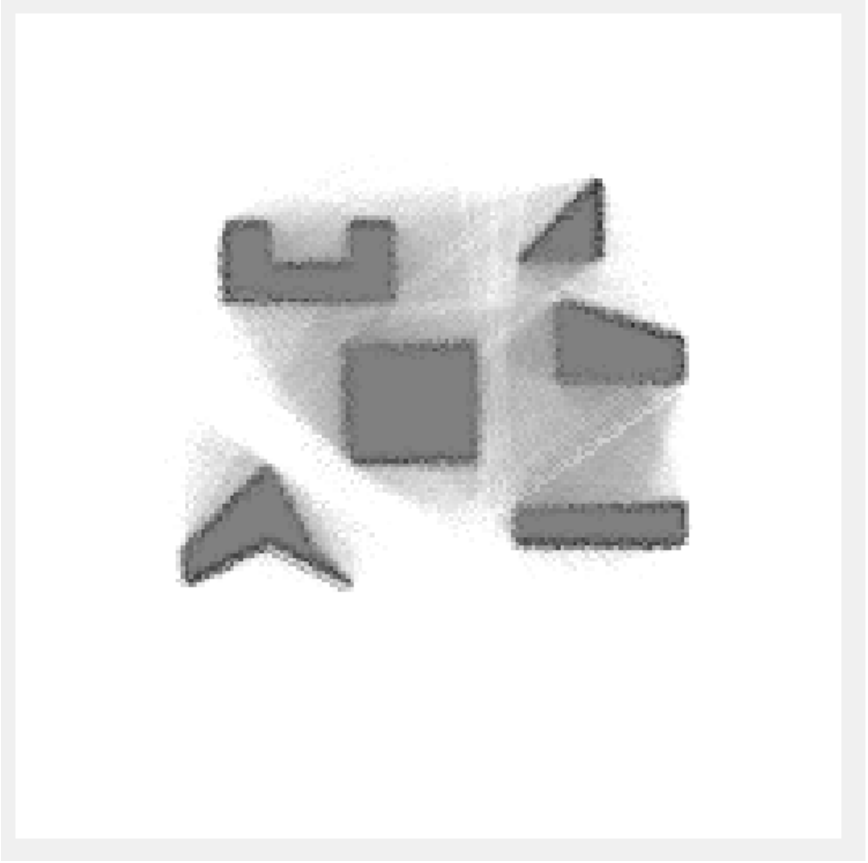


Figura A.94: Mapa resultante de *MappingTB_v10_1(0.8,0.1)*

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0	1	0	0.0291
3	0	2	0	0.03
4	0	3	0	0.0286
5	0	4	0	0.0199
6	0	5	0	0.0228
7	0	6	0	0.0357
8	0	7	0	0.0617
9	0	8	0	0.075
10	0	9	0	0.0665
11	0	10	0	0.0462
12	0	11	0	0.0524
13	0	12	0	0.0726
14	0	13	0	0.0674
15	0.179	14.04	-0.17	0.0766
16	0.532	15.04	-0.34	0.095
17	1.049	15.96	-0.51	0.0779
18	1.714	16.79	-0.68	0.104
19	2.509	17.49	-0.85	0.0932
20	3.411	18.04	-1.02	0.0816
21	4.394	18.43	-1.19	0.1107
22	5.428	18.66	-1.36	0.1156
23	6.486	18.7	-1.53	0.126
24	7.535	18.56	-1.7	0.1151
25	8.527	18.43	-1.7	0.1115
26	9.519	18.3	-1.7	0.0982
27	10.51	18.18	-1.7	0.0898
28	11.5	18.05	-1.7	0.1007
29	12.49	17.92	-1.7	0.1181
30	13.48	17.79	-1.7	0.1245
31	14.48	17.66	-1.7	0.1319
32	15.47	17.53	-1.7	0.1621
33	16.46	17.4	-1.7	0.1899
34	17.45	17.27	-1.7	0.2331
35	17.59	17.23	-1.88	0.2253
36	17.73	17.16	-2.06	0.2121
37	17.84	17.06	-2.24	0.1985
38	17.94	16.95	-2.42	0.1893
39	18.02	16.82	-2.6	0.194
40	18.07	16.68	-2.78	0.1887
41	18.1	16.54	-2.96	0.1862
42	18.1	16.39	-3.14	0.184
43	18.1	15.39	-3.14	0.1837
44	18.11	14.39	-3.14	0.2142
45	18.11	13.39	-3.14	0.196

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
46	18.11	12.39	-3.14	0.1817
47	18.11	11.39	-3.14	0.2275
48	18.11	10.39	-3.14	0.2413
49	18.11	9.385	-3.14	0.2402
50	18.11	8.385	-3.14	0.258
51	18.12	7.385	-3.14	0.2598
52	18.12	6.385	-3.14	0.2428
53	18.12	5.385	-3.14	0.2478
54	18.12	4.385	-3.14	0.2641
55	18.12	3.385	-3.14	0.2828
56	18.12	2.385	-3.14	0.2819
57	18.13	1.385	-3.14	0.2697
58	18.13	1.385	-2.48	0.2704
59	18.13	1.385	-1.82	0.2571
60	18.13	1.385	-1.16	0.2673
61	18.13	1.385	-0.5	0.2667
62	18.13	1.385	0.16	0.2753
63	18.13	1.385	0.82	0.282
64	18.13	1.385	1.48	0.2833
65	17.3	1.46	1.48	0.2792
66	16.47	1.536	1.48	0.2539
67	15.64	1.611	1.48	0.2395
68	14.81	1.687	1.48	0.2332
69	13.98	1.763	1.48	0.2215
70	13.15	1.838	1.48	0.2246
71	12.32	1.914	1.48	0.2297
72	11.49	1.989	1.48	0.2158
73	10.66	2.065	1.48	0.223
74	9.827	2.14	1.48	0.2217
75	8.997	2.216	1.48	0.2401
76	8.167	2.291	1.48	0.2578
77	7.337	2.367	1.48	0.2553
78	6.507	2.443	1.48	0.2501
79	5.677	2.518	1.48	0.2651
80	4.847	2.594	1.48	0.2592
81	4.017	2.669	1.48	0.2675
82	3.188	2.745	1.48	0.2883
83	2.358	2.82	1.48	0.2993
84	1.528	2.896	1.48	0.3097
Error cuadrático medio				0.1794

Tabla A.20: Error de pose en *MappingTB_v10_1*(0.8,0.1)

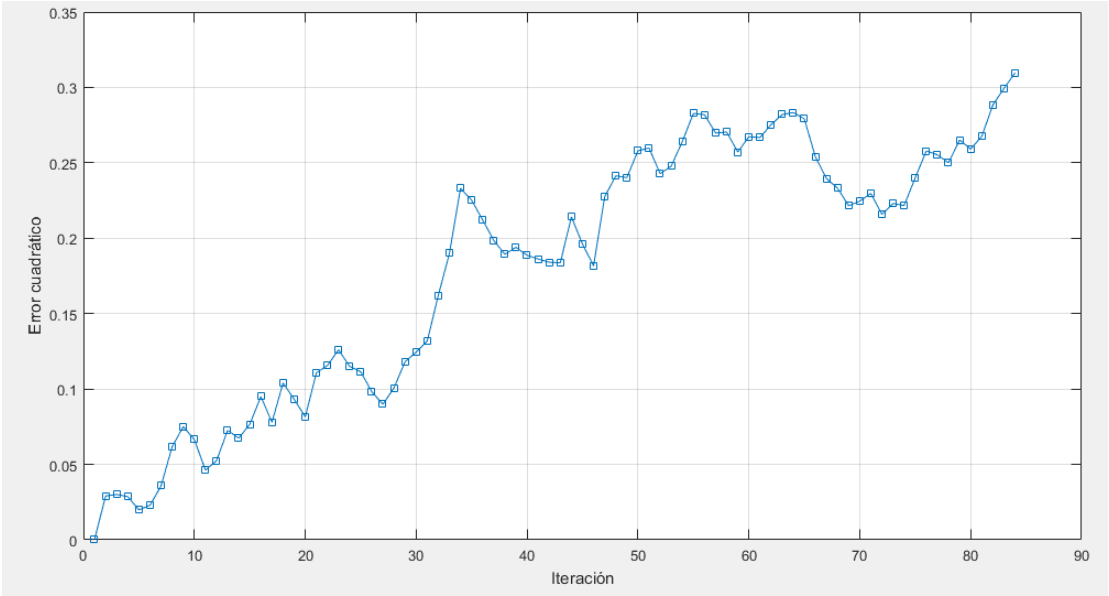


Figura A.95: Gráfica del error de pose en *MappingTB_v10_1*(0.8,0.1)

A.19. MappingTB_v10_2

```
1 MappingTB_v10_2(0.16,0.05)
```

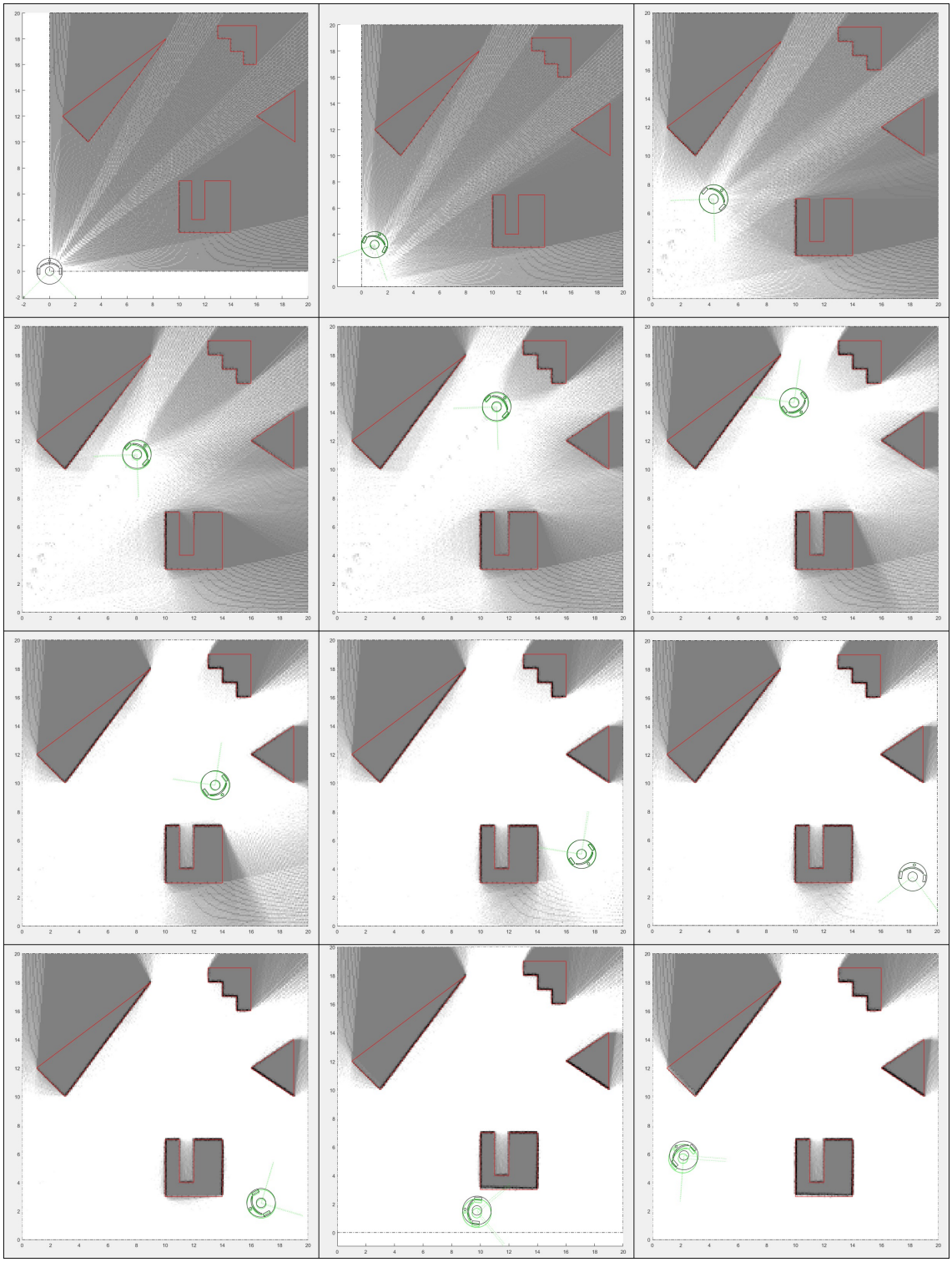


Figura A.96: Resultado de *MappingTB_v10_2* con errores de odometría 0.16 y de distancia 0.05. Obstáculos en rojo, mapa en escala de grises y pose en negro.

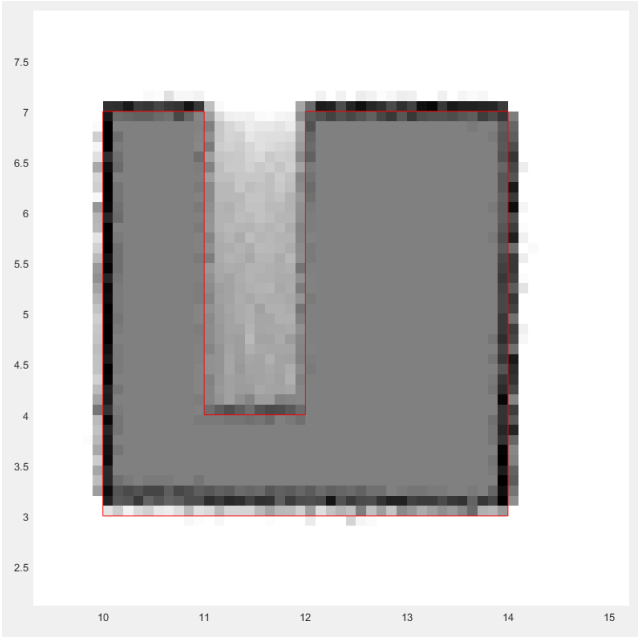


Figura A.97: Ampliación del resultado final de *MappingTB_v10_2* con errores de odometría 0.16 y de distancia 0.05

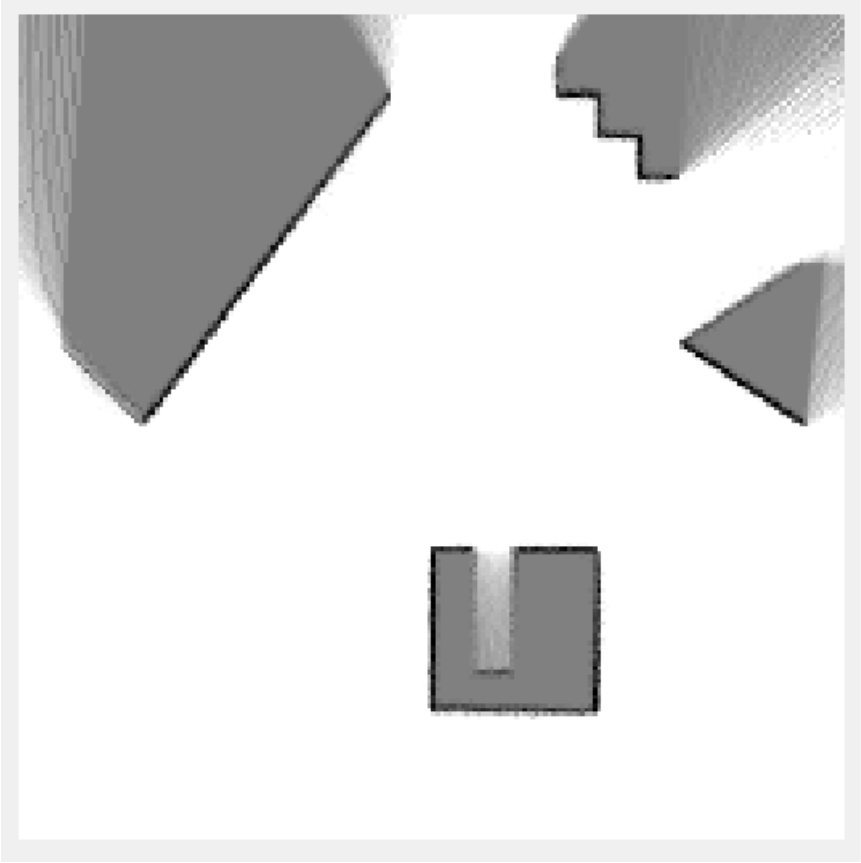


Figura A.98: Mapa resultante de *MappingTB_v10_2(0.16,0.05)*

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0.168	1.112	-0.15	0.0381
3	0.501	2.187	-0.3	0.0405
4	0.99	3.2	-0.45	0.0274
5	1.625	4.129	-0.6	0.0327
6	2.392	4.952	-0.75	0.0135
7	3.017	5.622	-0.75	0.0155
8	3.642	6.293	-0.75	0.0511
9	4.266	6.964	-0.75	0.0189
10	4.891	7.635	-0.75	0.0339
11	5.516	8.305	-0.75	0.0419
12	6.141	8.976	-0.75	0.0697
13	6.766	9.647	-0.75	0.0442
14	7.391	10.32	-0.75	0.0505
15	8.016	10.99	-0.75	0.0802
16	8.64	11.66	-0.75	0.0748
17	9.265	12.33	-0.75	0.0822
18	9.89	13	-0.75	0.0732
19	10.52	13.67	-0.75	0.0629
20	11.14	14.34	-0.75	0.0667
21	10.88	14.21	-1.1	0.0646
22	10.59	14.17	-1.45	0.0589
23	10.31	14.24	-1.8	0.0741
24	10.06	14.4	-2.15	0.061
25	9.888	14.63	-2.5	0.0634
26	10.49	13.83	-2.5	0.0705
27	11.08	13.03	-2.5	0.0582
28	11.68	12.23	-2.5	0.0475
29	12.28	11.43	-2.5	0.0403
30	12.88	10.63	-2.5	0.0377
31	13.48	9.827	-2.5	0.0532
32	14.08	9.026	-2.5	0.0553
33	14.68	8.225	-2.5	0.0606
34	15.27	7.424	-2.5	0.0747
35	15.87	6.622	-2.5	0.0716
36	16.47	5.821	-2.5	0.0592
37	17.07	5.02	-2.5	0.0363
38	17.67	4.219	-2.5	0.0218
39	18.27	3.418	-2.5	0.0368
40	18.27	3.418	-1.91	0.0241
41	18.27	3.418	-1.32	0.0242
42	18.27	3.418	-0.73	0.019
43	18.27	3.418	-0.14	0.0119
44	18.27	3.418	0.45	0.0077
45	18.27	3.418	1.04	0.0194

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
46	18.27	3.418	1.63	0.031
47	18.27	3.418	2.22	0.0514
48	17.51	2.931	2.145	0.0398
49	16.73	2.502	2.07	0.0785
50	15.91	2.134	1.995	0.0871
51	15.07	1.827	1.92	0.0877
52	14.21	1.585	1.845	0.1043
53	13.33	1.407	1.77	0.1268
54	12.44	1.296	1.695	0.1485
55	11.55	1.252	1.62	0.1957
56	10.65	1.275	1.545	0.2074
57	9.758	1.366	1.47	0.1903
58	8.876	1.522	1.395	0.1803
59	8.008	1.745	1.32	0.1886
60	7.16	2.031	1.245	0.1864
61	6.335	2.381	1.17	0.1808
62	5.539	2.791	1.095	0.183
63	4.775	3.26	1.02	0.1917
64	4.049	3.785	0.945	0.1879
65	3.364	4.362	0.87	0.1913
66	2.725	4.99	0.795	0.2081
67	2.134	5.663	0.72	0.2343
Error cuadrático medio				0.0799

Tabla A.21: Error de pose en *MappingTB_v10_2*(0.16,0.05)

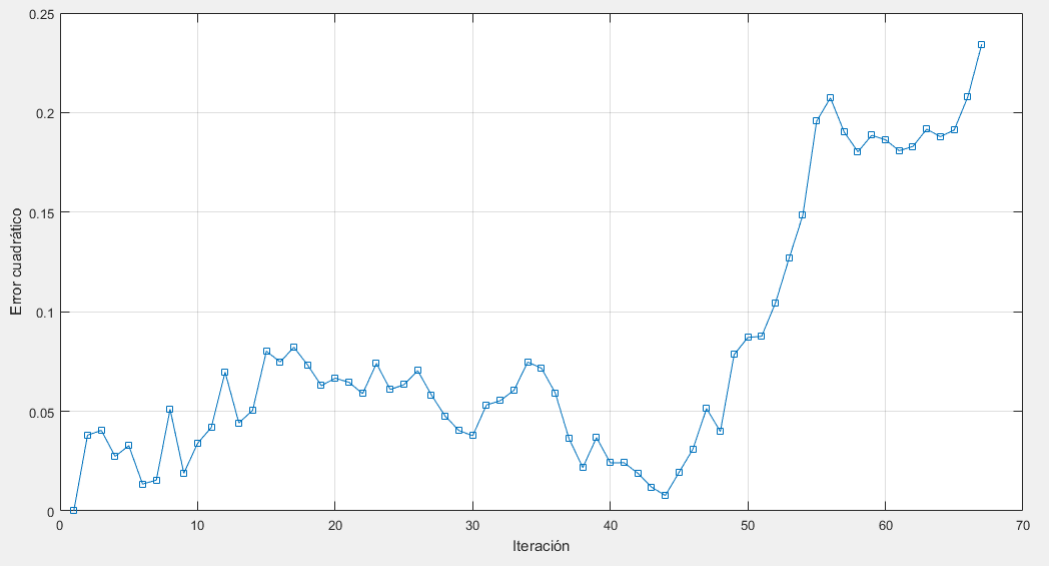


Figura A.99: Gráfica del error de pose en *MappingTB_v10_2*(0.16,0.05)

```
1 MappingTB_v10_2(0.3,0.1)
```

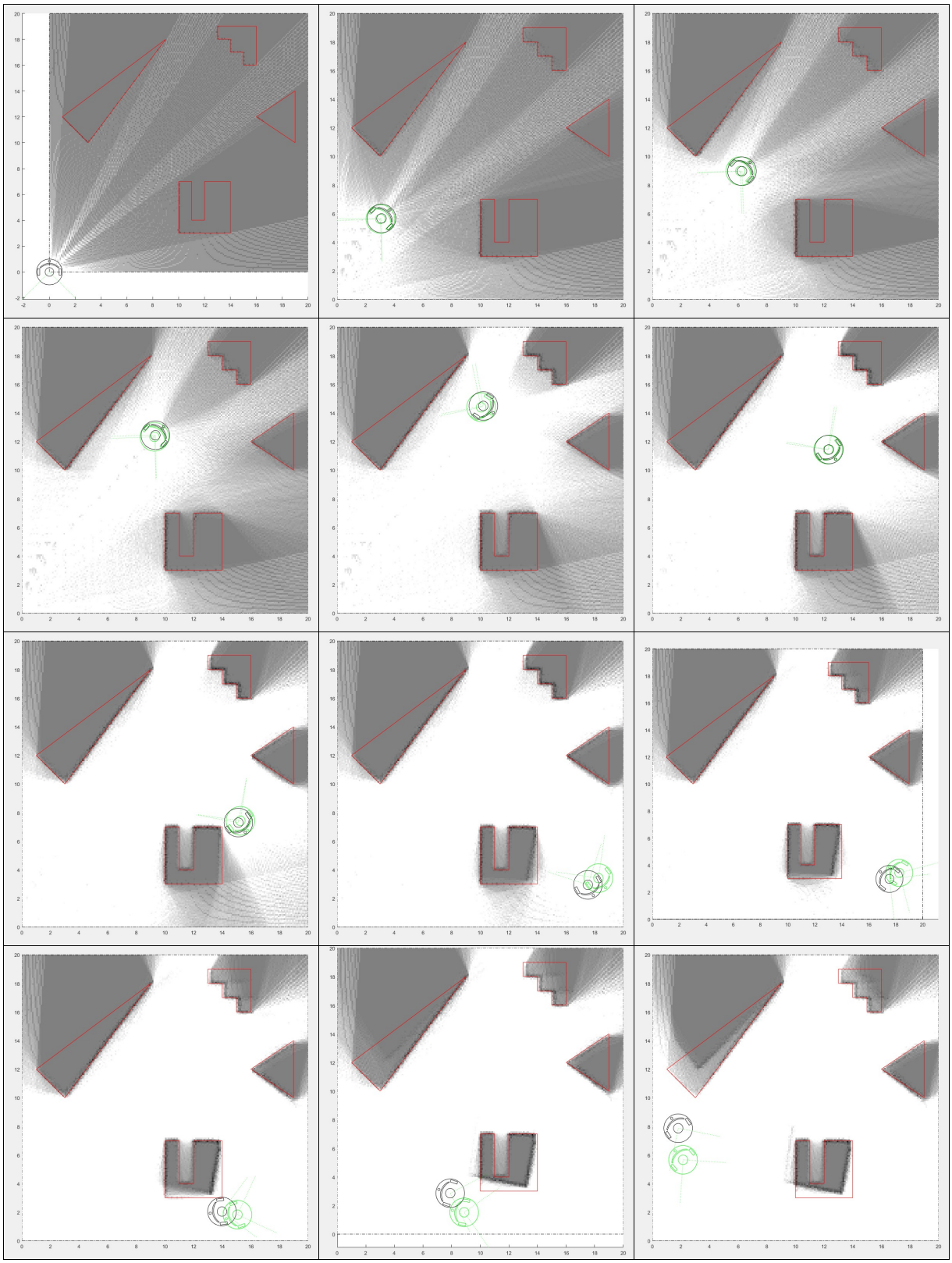


Figura A.100: Resultado de *MappingTB_v10_2* con errores de odometría 0.3 y de distancia 0.1. Obstáculos en rojo, mapa en escala de grises y pose en negro.

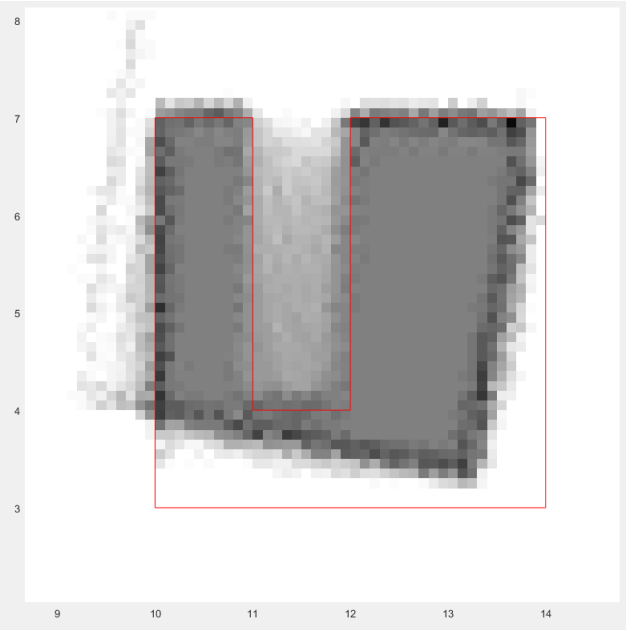


Figura A.101: Ampliación del resultado final de *MappingTB_v10_2* con errores de odometría 0.3 y de distancia 0.1

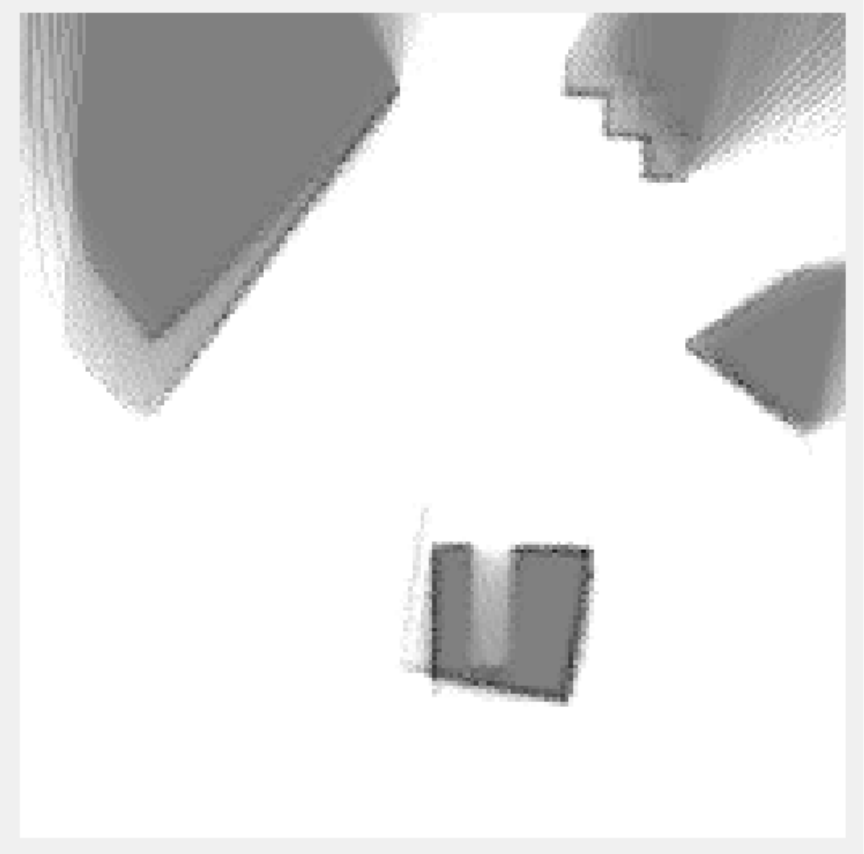


Figura A.102: Mapa resultante de *MappingTB_v10_2(0.3,0.1)*

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0.168	1.112	-0.15	0.0396
3	0.501	2.187	-0.3	0.0812
4	0.99	3.2	-0.45	0.0712
5	1.625	4.129	-0.6	0.0877
6	2.392	4.952	-0.75	0.0734
7	3.017	5.622	-0.75	0.0912
8	3.642	6.293	-0.75	0.0595
9	4.266	6.964	-0.75	0.0642
10	4.891	7.635	-0.75	0.0497
11	5.516	8.305	-0.75	0.1502
12	6.141	8.976	-0.75	0.118
13	6.766	9.647	-0.75	0.1055
14	7.391	10.32	-0.75	0.1387
15	8.016	10.99	-0.75	0.1448
16	8.64	11.66	-0.75	0.1377
17	9.265	12.33	-0.75	0.1328
18	9.89	13	-0.75	0.1338
19	10.52	13.67	-0.75	0.1298
20	11.14	14.34	-0.75	0.1524
21	10.88	14.21	-1.1	0.1368
22	10.59	14.17	-1.45	0.1825
23	10.31	14.24	-1.8	0.167
24	10.06	14.4	-2.15	0.1807
25	9.888	14.63	-2.5	0.1829
26	10.49	13.83	-2.5	0.1574
27	11.08	13.03	-2.5	0.1319
28	11.68	12.23	-2.5	0.1252
29	12.28	11.43	-2.5	0.0642
30	12.88	10.63	-2.5	0.0571
31	13.48	9.827	-2.5	0.0617
32	14.08	9.026	-2.5	0.1066
33	14.68	8.225	-2.5	0.1842
34	15.27	7.424	-2.5	0.1923
35	15.87	6.622	-2.5	0.231
36	16.47	5.821	-2.5	0.3505
37	17.07	5.02	-2.5	0.5357
38	17.67	4.219	-2.5	0.7046
39	18.27	3.418	-2.5	0.8923
40	18.27	3.418	-1.91	0.8674
41	18.27	3.418	-1.32	0.8605
42	18.27	3.418	-0.73	0.8515
43	18.27	3.418	-0.14	0.8491
44	18.27	3.418	0.45	0.8479
45	18.27	3.418	1.04	0.8641

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
46	18.27	3.418	1.63	0.8913
47	18.27	3.418	2.22	0.8828
48	17.51	2.931	2.145	0.9602
49	16.73	2.502	2.07	1.042
50	15.91	2.134	1.995	1.038
51	15.07	1.827	1.92	1.123
52	14.21	1.585	1.845	1.157
53	13.33	1.407	1.77	1.22
54	12.44	1.296	1.695	1.302
55	11.55	1.252	1.62	1.425
56	10.65	1.275	1.545	1.495
57	9.758	1.366	1.47	1.552
58	8.876	1.522	1.395	1.66
59	8.008	1.745	1.32	1.744
60	7.16	2.031	1.245	1.831
61	6.335	2.381	1.17	1.881
62	5.539	2.791	1.095	1.913
63	4.775	3.26	1.02	1.991
64	4.049	3.785	0.945	2.019
65	3.364	4.362	0.87	2.086
66	2.725	4.99	0.795	2.156
67	2.134	5.663	0.72	2.237
Error cuadrático medio				0.6769

Tabla A.22: Error de pose en *MappingTB_v10_2*(0.3,0.1)

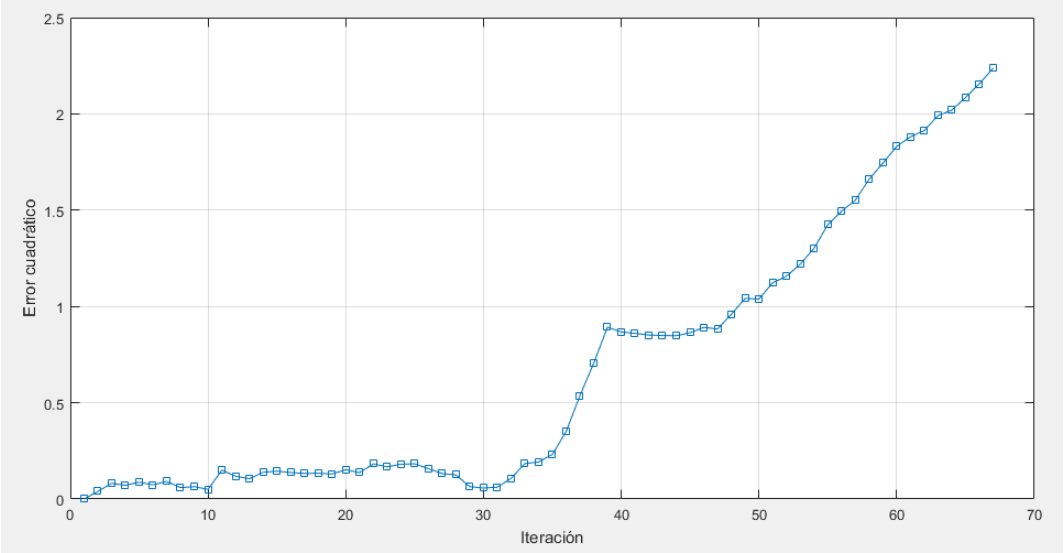


Figura A.103: Gráfica del error de pose en *MappingTB_v10_2*(0.3,0.1)

A.20. MappingTB_v10_3

```
1 MappingTB_v10_3(0.16,0.05)
```

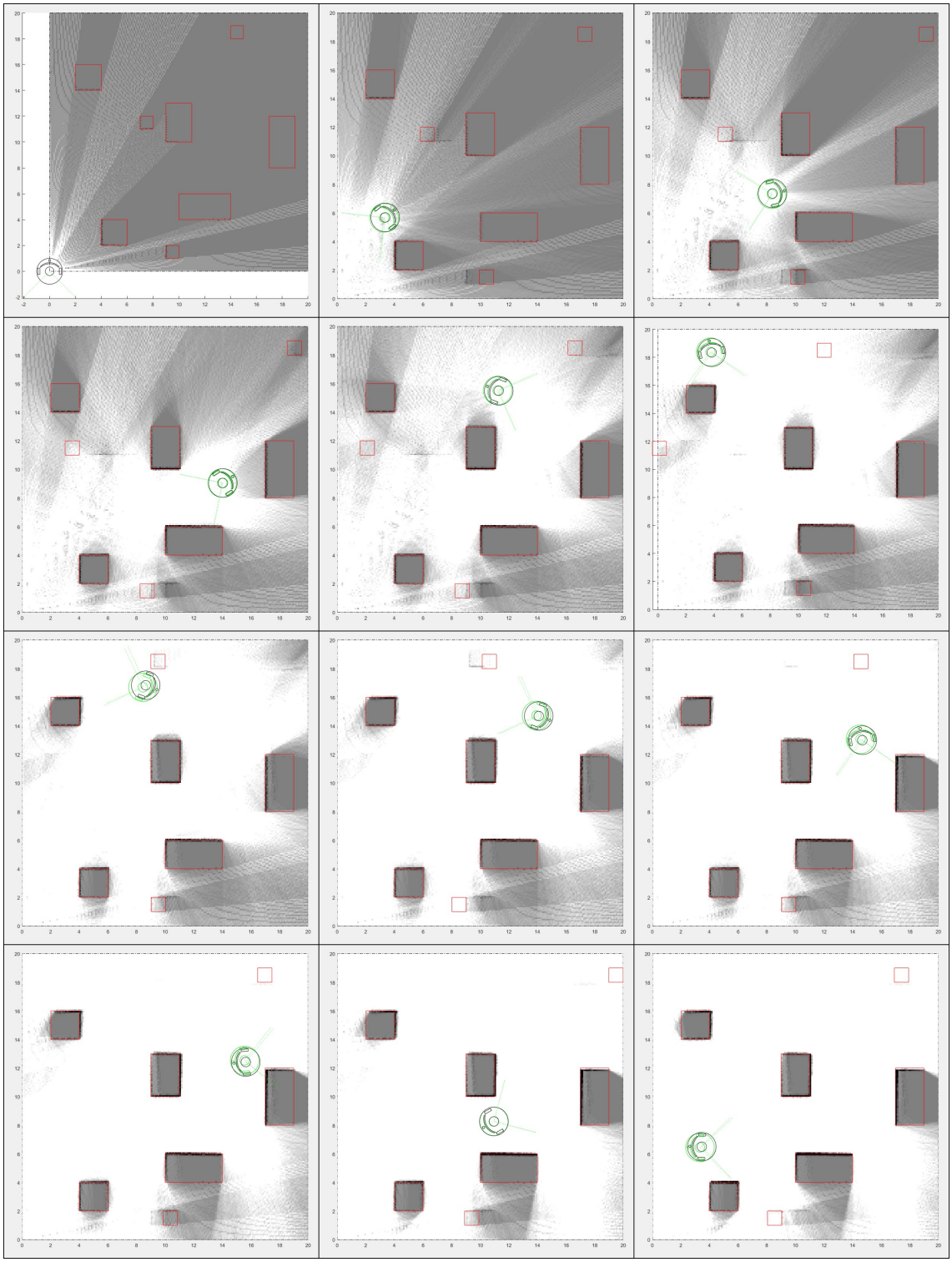


Figura A.104: Resultado de *MappingTB_v10_3* con errores de odometría 0.16 y de distancia 0.05. Obstáculos en rojo, mapa en escala de grises y pose en negro.

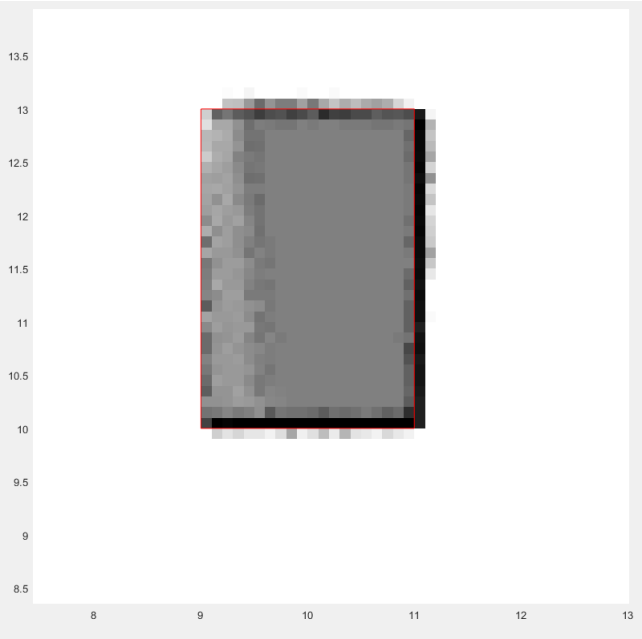


Figura A.105: Ampliación del resultado final de *MappingTB_v10_3* con errores de odometría 0.16 y de distancia 0.05

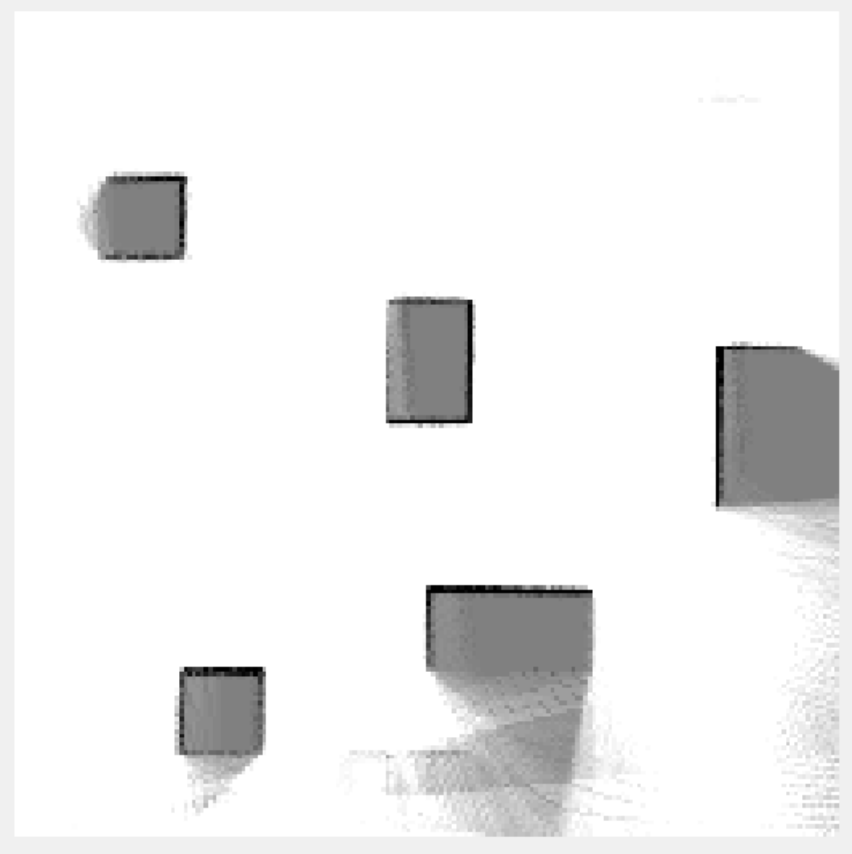


Figura A.106: Mapa resultante de *MappingTB_v10_3(0.16,0.05)*

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0.168	1.112	-0.15	0.0054
3	0.501	2.187	-0.3	0.0279
4	0.99	3.2	-0.45	0.0266
5	1.625	4.129	-0.6	0.0743
6	2.392	4.952	-0.75	0.0631
7	3.273	5.651	-0.9	0.0845
8	4.249	6.211	-1.05	0.1029
9	5.298	6.619	-1.2	0.0802
10	6.395	6.865	-1.35	0.0601
11	7.046	7.011	-1.35	0.0704
12	7.696	7.157	-1.35	0.0779
13	8.347	7.303	-1.35	0.081
14	8.997	7.449	-1.35	0.0779
15	9.648	7.595	-1.35	0.0711
16	10.3	7.741	-1.35	0.0531
17	10.95	7.887	-1.35	0.045
18	11.6	8.033	-1.35	0.0467
19	12.25	8.179	-1.35	0.0487
20	12.9	8.325	-1.35	0.057
21	13.99	9.046	-0.985	0.0538
22	14.74	10.11	-0.62	0.1007
23	15.07	11.37	-0.255	0.1173
24	14.93	12.67	0.11	0.1024
25	14.33	13.82	0.475	0.1072
26	13.36	14.7	0.84	0.0958
27	12.15	15.16	1.205	0.0997
28	11.21	15.52	1.205	0.0845
29	10.28	15.88	1.205	0.0905
30	9.343	16.24	1.205	0.0911
31	8.41	16.59	1.205	0.1015
32	7.476	16.95	1.205	0.1053
33	6.542	17.31	1.205	0.0988
34	5.608	17.67	1.205	0.0822
35	4.674	18.02	1.205	0.0768
36	3.74	18.38	1.205	0.0937
37	3.74	18.38	0.685	0.1039
38	3.74	18.38	0.165	0.107
39	3.74	18.38	-0.355	0.1148
40	3.74	18.38	-0.875	0.1202
41	3.74	18.38	-1.395	0.1352
42	3.74	18.38	-1.915	0.1333
43	4.525	18.1	-1.915	0.1368
44	5.309	17.82	-1.915	0.1773
45	6.094	17.54	-1.915	0.1999

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
46	6.878	17.26	-1.915	0.229
47	7.663	16.98	-1.915	0.2261
48	8.447	16.69	-1.915	0.242
49	9.232	16.41	-1.915	0.2403
50	10.02	16.13	-1.915	0.2279
51	10.8	15.85	-1.915	0.2122
52	11.59	15.57	-1.915	0.216
53	12.37	15.29	-1.915	0.1883
54	13.15	15.01	-1.915	0.1804
55	13.94	14.73	-1.915	0.1602
56	14.72	14.45	-1.915	0.1471
57	15.51	14.16	-1.915	0.1526
58	15.26	14.18	-1.615	0.1683
59	15.02	14.11	-1.315	0.1732
60	14.8	13.98	-1.015	0.181
61	14.64	13.79	-0.715	0.1693
62	14.54	13.56	-0.415	0.1599
63	14.51	13.31	-0.115	0.1551
64	14.56	13.07	0.185	0.1501
65	14.67	12.85	0.485	0.1542
66	14.85	12.67	0.785	0.1489
67	15.07	12.55	1.085	0.1482
68	15.32	12.51	1.385	0.1309
69	15.56	12.54	1.685	0.1399
70	15.79	12.64	1.985	0.1468
71	15.98	12.8	2.285	0.1402
72	16.11	13.01	2.585	0.1309
73	15.6	12.29	2.525	0.1089
74	15.05	11.6	2.465	0.1145
75	14.46	10.95	2.405	0.1003
76	13.82	10.33	2.345	0.0675
77	13.16	9.752	2.285	0.0342
78	12.46	9.215	2.225	0.0262
79	11.73	8.72	2.165	0.0108
80	10.97	8.27	2.105	0.0183
81	10.18	7.867	2.045	0.0212
82	9.37	7.511	1.985	0.0447
83	8.542	7.205	1.925	0.0614
84	7.696	6.949	1.865	0.0394
85	6.837	6.744	1.805	0.0651
86	5.967	6.591	1.745	0.0843
87	5.089	6.49	1.685	0.1069
Error cuadrático medio				0.1104

Tabla A.23: Error de pose en *MappingTB_v10_3*(0.16,0.05)

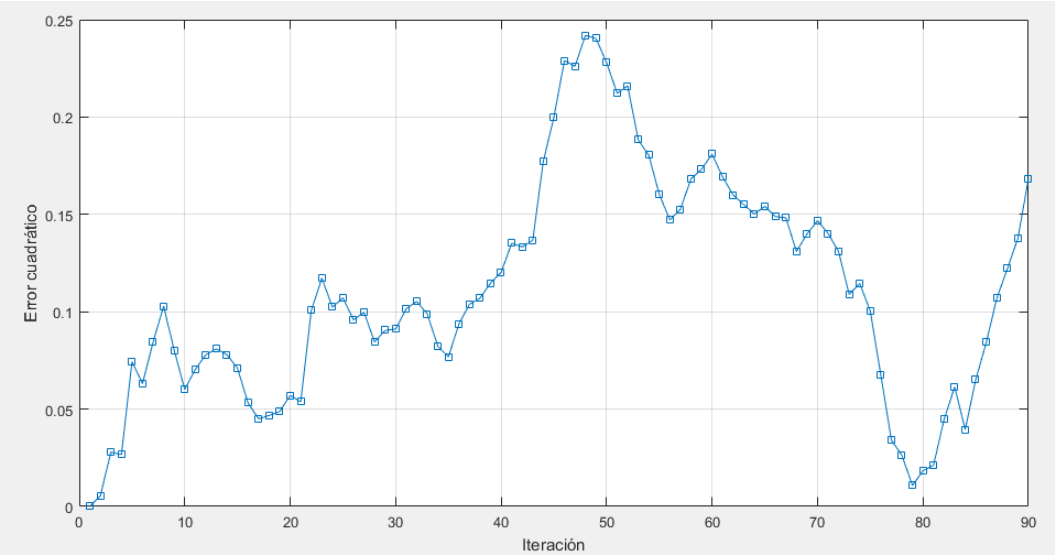


Figura A.107: Gráfica del error de pose en *MappingTB_v10_3*(0.16,0.05)

Profile Summary

Generated 06-Sep-2016 07:22:59 using performance time.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
MappingTB_v10_3	1	61.225 s	9.139 s	
VissionSimulation_v1	90	42.653 s	1.109 s	
Mapping_v9	90	5.785 s	0.417 s	
CalculateMAP_v3	94	4.899 s	3.398 s	
mapshow	90	1.241 s	0.021 s	
DisplayMap	1	1.017 s	0.127 s	
dibujar_v2	180	0.634 s	0.534 s	
CalculateMAP_v2	2403	0.351 s	0.351 s	
mean2	2403	0.119 s	0.119 s	

Figura A.108: Desempeño temporal de *MappingTB_v10_3(0.16,0.05)*

```
1 MappingTB_v10_3(0.2,0.1)
```

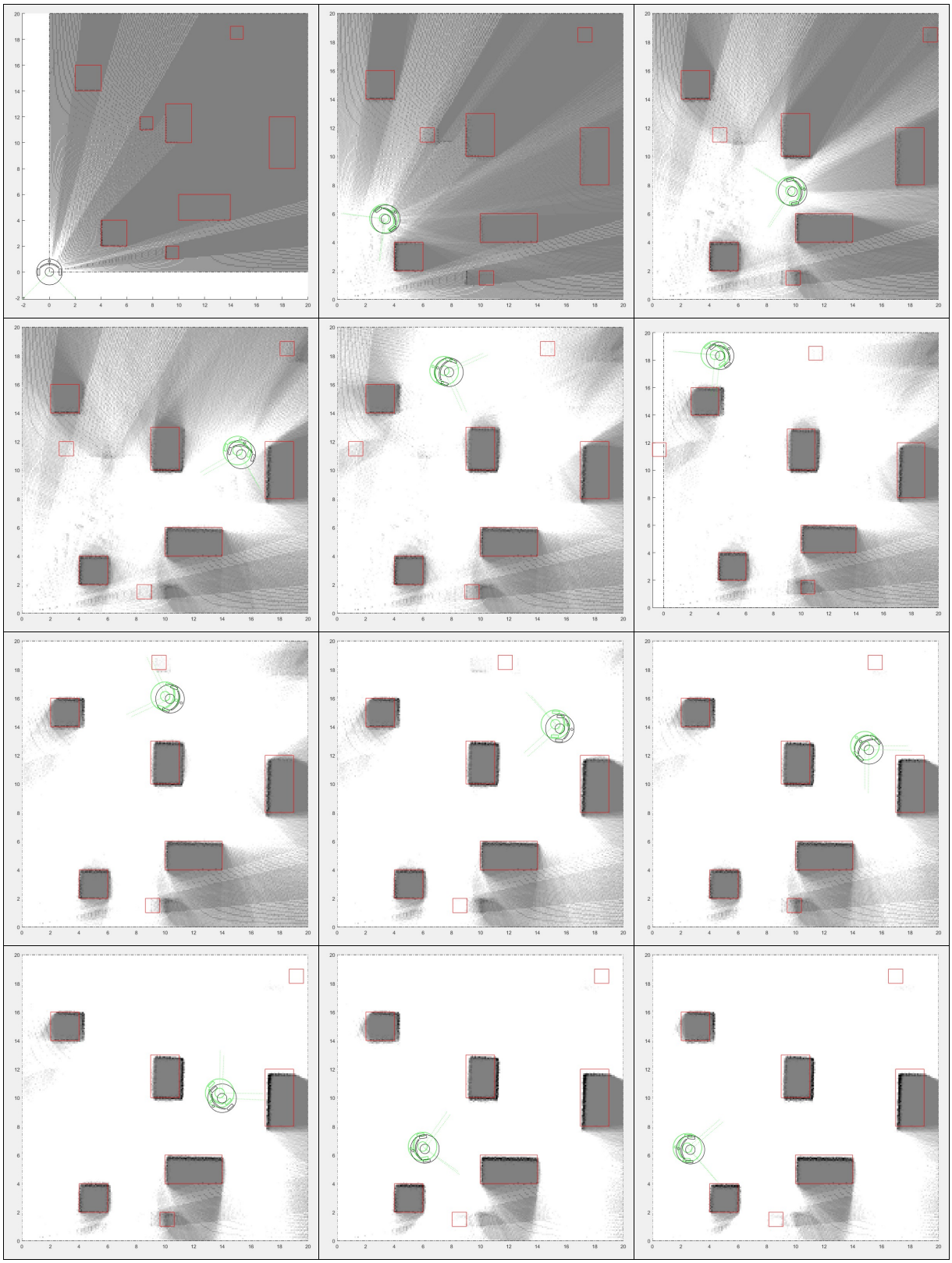


Figura A.109: Resultado de *MappingTB_v10_3* con errores de odometría 0.2 y de distancia 0.1. Obstáculos en rojo, mapa en escala de grises y pose en negro.

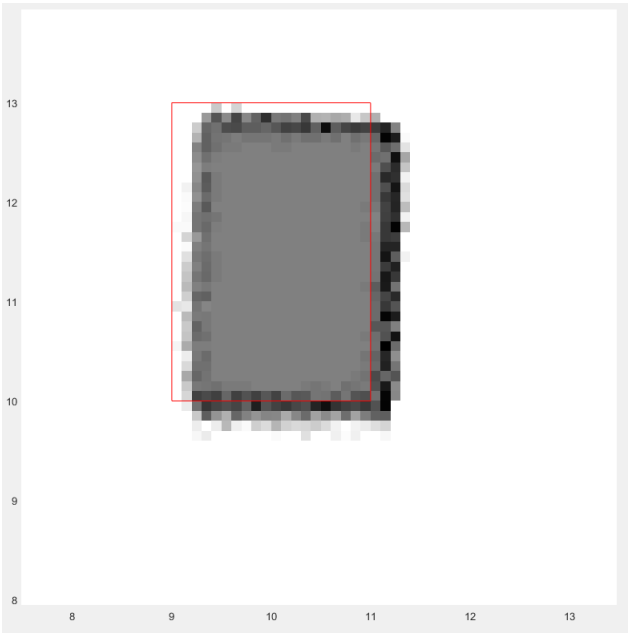


Figura A.110: Ampliación del resultado final de *MappingTB_v10_3* con errores de odometría 0.2 y de distancia 0.1

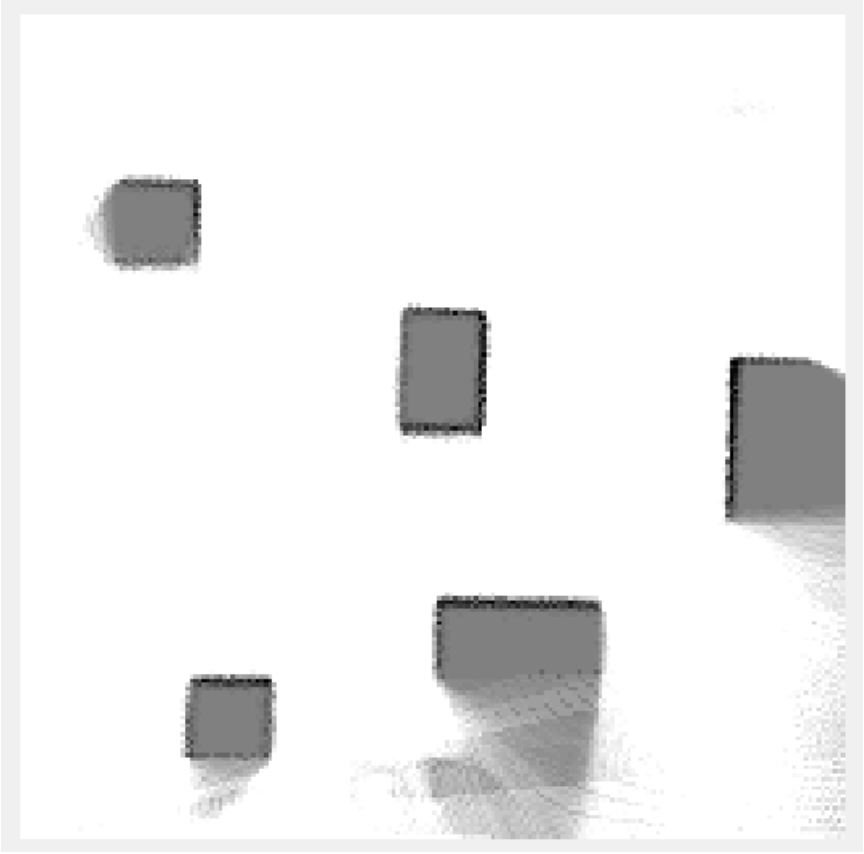


Figura A.111: Mapa resultante de *MappingTB_v10_3(0.2,0.1)*

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
1	0	0	0	0
2	0.168	1.112	-0.15	0.0228
3	0.501	2.187	-0.3	0.0626
4	0.99	3.2	-0.45	0.0979
5	1.625	4.129	-0.6	0.1338
6	2.392	4.952	-0.75	0.1152
7	3.273	5.651	-0.9	0.1347
8	4.249	6.211	-1.05	0.1455
9	5.298	6.619	-1.2	0.1548
10	6.395	6.865	-1.35	0.117
11	7.046	7.011	-1.35	0.1396
12	7.696	7.157	-1.35	0.1634
13	8.347	7.303	-1.35	0.1557
14	8.997	7.449	-1.35	0.1892
15	9.648	7.595	-1.35	0.1643
16	10.3	7.741	-1.35	0.1954
17	10.95	7.887	-1.35	0.2104
18	11.6	8.033	-1.35	0.2026
19	12.25	8.179	-1.35	0.2243
20	12.9	8.325	-1.35	0.2634
21	13.99	9.046	-0.985	0.2918
22	14.74	10.11	-0.62	0.325
23	15.07	11.37	-0.255	0.3753
24	14.93	12.67	0.11	0.3682
25	14.33	13.82	0.475	0.3952
26	13.36	14.7	0.84	0.3983
27	12.15	15.16	1.205	0.3902
28	11.21	15.52	1.205	0.3945
29	10.28	15.88	1.205	0.3896
30	9.343	16.24	1.205	0.365
31	8.41	16.59	1.205	0.3613
32	7.476	16.95	1.205	0.3513
33	6.542	17.31	1.205	0.356
34	5.608	17.67	1.205	0.3676
35	4.674	18.02	1.205	0.3675
36	3.74	18.38	1.205	0.3836
37	3.74	18.38	0.685	0.3926
38	3.74	18.38	0.165	0.3866
39	3.74	18.38	-0.355	0.3819
40	3.74	18.38	-0.875	0.3823
41	3.74	18.38	-1.395	0.3795
42	3.74	18.38	-1.915	0.3806
43	4.525	18.1	-1.915	0.3486
44	5.309	17.82	-1.915	0.3583
45	6.094	17.54	-1.915	0.3419

Sigue en la página siguiente

Iteración	Pose X	Pose Y	Pose THETA	Error de pose
46	6.878	17.26	-1.915	0.3273
47	7.663	16.98	-1.915	0.3307
48	8.447	16.69	-1.915	0.3305
49	9.232	16.41	-1.915	0.3596
50	10.02	16.13	-1.915	0.3812
51	10.8	15.85	-1.915	0.3903
52	11.59	15.57	-1.915	0.3904
53	12.37	15.29	-1.915	0.4088
54	13.15	15.01	-1.915	0.3969
55	13.94	14.73	-1.915	0.4163
56	14.72	14.45	-1.915	0.4329
57	15.51	14.16	-1.915	0.4331
58	15.26	14.18	-1.615	0.4308
59	15.02	14.11	-1.315	0.434
60	14.8	13.98	-1.015	0.4437
61	14.64	13.79	-0.715	0.4322
62	14.54	13.56	-0.415	0.4373
63	14.51	13.31	-0.115	0.4388
64	14.56	13.07	0.185	0.4204
65	14.67	12.85	0.485	0.4301
66	14.85	12.67	0.785	0.4016
67	15.07	12.55	1.085	0.4067
68	15.32	12.51	1.385	0.3918
69	15.56	12.54	1.685	0.4035
70	15.79	12.64	1.985	0.4152
71	15.98	12.8	2.285	0.4361
72	16.11	13.01	2.585	0.441
73	15.6	12.29	2.525	0.4393
74	15.05	11.6	2.465	0.3908
75	14.46	10.95	2.405	0.3991
76	13.82	10.33	2.345	0.3989
77	13.16	9.752	2.285	0.3547
78	12.46	9.215	2.225	0.356
79	11.73	8.72	2.165	0.3196
80	10.97	8.27	2.105	0.2858
81	10.18	7.867	2.045	0.279
82	9.37	7.511	1.985	0.2962
83	8.542	7.205	1.925	0.2678
84	7.696	6.949	1.865	0.2757
85	6.837	6.744	1.805	0.2431
86	5.967	6.591	1.745	0.233
87	5.089	6.49	1.685	0.2636
Error cuadrático medio				0.319

Tabla A.24: Error de pose en *MappingTB_v10_3*(0.2,0.1)

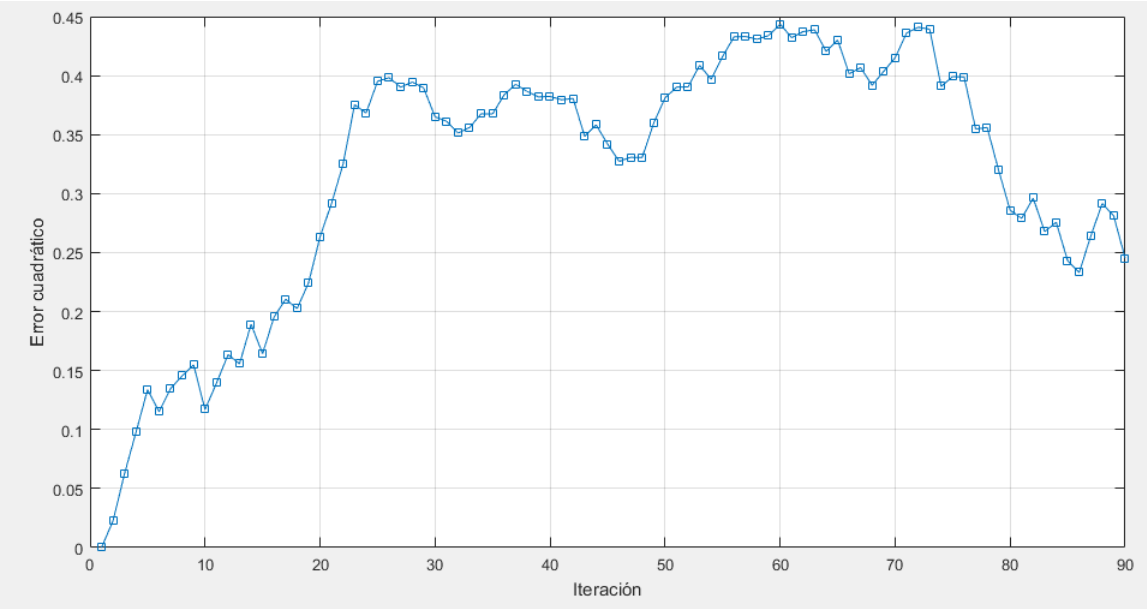


Figura A.112: Gráfica del error de pose en *MappingTB_v10_3*(0.2,0.1)

Se han realizado varias simulaciones de *MappingTB_v10_3(0.16,0.05)* para tener una idea de cuantas veces se corrige adecuadamente la posición. A continuación los resultados.

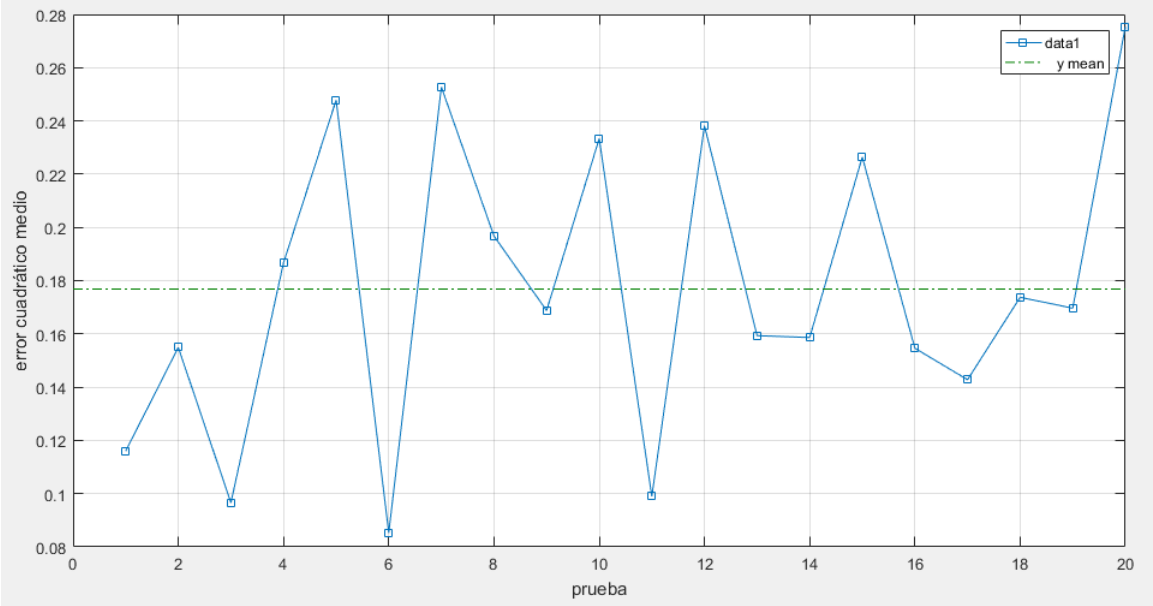


Figura A.113: Gráfica de errores cuadráticos medios para múltiples simulaciones de *MappingTB_v10_3(0.16,0.05)*

Apéndice B

Códigos

B.1. Mapping

B.1.1. Mapping_v1

```
function [MAPX,MAPY] = Mapping_v1(x,y,theta,RAD)

    AngInicial = 225;
    Paso = 0.5;                          %valor positivo
    AngFinal = -45;

    NumMedidas = abs((AngInicial - AngFinal)/Paso) + 1;

    if AngInicial > AngFinal
        Paso = -Paso;
    end

    if length(RAD) ~= NumMedidas
        error('dimensiones incorrectas del vector de medidas');
    end

    ANG = AngInicial : Paso : AngFinal; %define vector de angulos

    MapPartX = zeros(1,NumMedidas);
    MapPartY = zeros(1,NumMedidas);

    for i = 1 : NumMedidas          %cambio de coordendas polares a cartesianas
            MapPartX(i) = RAD(i) * cos(ANG(i)*pi/180);
            MapPartY(i) = RAD(i) * sin(ANG(i)*pi/180);
    end

    MapPart = [ MapPartX;         %unifica y da formato a los puntos para ser convertidos
                MapPartY;
                ones(size(MapPartX))];

    T01 = [ cos(theta)  -sin(theta)       x;
            sin(theta)   cos(theta)       y;
            0            0                1 ];

    MAP = T01 * MapPart;           %cambio de sistema de referencia

    MAPX = MAP(1,:);
    MAPY = MAP(2,:);

end
```

B.1.2. Mapping_v2

```
function [MAPX,MAPY] = Mapping_v2(RESET,R,T,wd,wi,RAD)

    AngInicial = 225;
    Paso = 0.5;                    %valor positivo
    AngFinal = -45;

    persistent xpersistent;
    persistent ypersistent;
    persistent thetapersistent;

    if RESET == 1                  %resetea a pose inicial
        xpersistent       = 0;
        ypersistent       = 0;
        thetapersistent   = 0;
        return;
    end

    r = R/6;                       %r(radio ruedas) R(radio Robot)

    Vd = wd*r;                     %cinemática
    Vi = wi*r;
    V = (Vd+Vi)/2;
    W = (Vd-Vi)/((10*R)/6);

    thetapersistent = thetapersistent + W*T;
    xpersistent = xpersistent + V*cos(thetapersistent + pi/2)*T;
    ypersistent = ypersistent + V*sin(thetapersistent + pi/2)*T;

    NumMedidas = abs((AngInicial - AngFinal)/Paso) + 1;

    if AngInicial > AngFinal                    %corrige signo del paso
        Paso = -Paso;
    end

    if length(RAD) ~= NumMedidas
        error('dimensiones incorrectas del vector de medidas');
    end

    ANG = AngInicial : Paso : AngFinal;                    %vector de angulos

    MapPartX = zeros(1,NumMedidas);                        %inicializamos vectores
    MapPartY = zeros(1,NumMedidas);

    for i = 1 : NumMedidas                                 %bucle de identificacion de
        distancias
            MapPartX(i) = RAD(i) * cos(ANG(i)*pi/180);
            MapPartY(i) = RAD(i) * sin(ANG(i)*pi/180);
    end

    MapPart = [ MapPartX;                                  %transformación de SR
                MapPartY;
                ones(size(MapPartX))];

    T01 = [ cos(thetapersistent)  -sin(thetapersistent)       xpersistent;
            sin(thetapersistent)   cos(thetapersistent)       ypersistent;
            0                      0                          1 ];

    MAP = T01 * MapPart;

    MAPX = MAP(1,:);
    MAPY = MAP(2,:);

end
```

B.1.3. Mapping_v3

```
function [MAPX,MAPY,x,y,theta] = Mapping_v3(RESET,R,T,wd,wi,RAD)

    AngInicial = 225;                   %grados sexagesimales
    Paso = 0.5;                         %valor positivo
    AngFinal = -45;

    r = R/6;                            %r(radio ruedas) R(radio Robot)

    persistent xpersistent;
    persistent ypersistent;
    persistent thetapersistent;

    if RESET == 1                       %pose inicial
        xpersistent       = 0;
        ypersistent       = 0;
        thetapersistent   = 0;
    else
        Vd = wd*r;                      %cinemática
        Vi = wi*r;
        V = (Vd+Vi)/2;
        W = (Vd-Vi)/((10*R)/6);

        thetapersistent = thetapersistent + W*T;
        xpersistent = xpersistent + V*cos(thetapersistent + pi/2)*T;
        ypersistent = ypersistent + V*sin(thetapersistent + pi/2)*T;
    end

    NumMedidas = abs((AngInicial - AngFinal)/Paso) + 1;

    if AngInicial > AngFinal            %corrige signo del paso
        Paso = -Paso;
    end

    if length(RAD) ~= NumMedidas
        error('dimensiones incorrectas del vector de medidas');
    end

    ANG = AngInicial : Paso : AngFinal;         %vector de angulos

    MapPartX = zeros(1,NumMedidas);             %inicializamos vectores
    MapPartY = zeros(1,NumMedidas);

    for i = 1 : NumMedidas                      %bucle de identificacion de distancias
            MapPartX(i) = RAD(i) * cos(ANG(i)*pi/180);
            MapPartY(i) = RAD(i) * sin(ANG(i)*pi/180);
    end

    MapPart = [ MapPartX;                       %transformación de SR
                MapPartY;
                ones(size(MapPartX))];

    T01 = [ cos(thetapersistent)  -sin(thetapersistent)       xpersistent;
            sin(thetapersistent)   cos(thetapersistent)       ypersistent;
            0                      0                          1 ];

    MAP = T01 * MapPart;

    MAPX = MAP(1,:);
    MAPY = MAP(2,:);

    x       = xpersistent;                  %valores de salida
    y       = ypersistent;
    theta   = thetapersistent;
end
```

B.1.4. Mapping_v4

```
function [MAP,x,y,theta] = Mapping_v4(RESET,LongRes,R,T,wd,wi,RAD)

    Incremento = 0.1;                   %longitud de la celda de posicion discretizada(metros)

    res = fix(LongRes / Incremento);    %dimension del mapa discreto en numero de celdas por
        lado

    r = R/6;                            %r(radio ruedas) R(radio Robot)

    persistent xpersistent;             %variables estáticas
    persistent ypersistent;
    persistent thetapersistent;
    persistent MAPpersistent;

    if RESET == 1                       %posición inicial
        xpersistent       = 0;
        ypersistent       = 0;
        thetapersistent   = 0;
        MAPpersistent     = zeros(res,res);
    else
        Vd = wd*r;                      %cinemática
        Vi = wi*r;
        V = (Vd+Vi)/2;
        W = (Vd-Vi)/((10*R)/6);

        thetapersistent = thetapersistent + W*T;
        xpersistent = xpersistent + V*cos(thetapersistent + pi/2)*T;
        ypersistent = ypersistent + V*sin(thetapersistent + pi/2)*T;
    end

    %obtiene mapa discreto
    newMAP = CalculateMAP_v1(LongRes,Incremento,xpersistent,ypersistent,thetapersistent,RAD);

    MAPpersistent = MAPpersistent | newMAP;    %agrega mapa nuevo al mapa global

    x       = xpersistent;                  %variables de salida
    y       = ypersistent;
    theta   = thetapersistent;
    MAP     = MAPpersistent;

end
```

B.1.5. Mapping_v5

```
function [MAP,x,y,theta] = Mapping_v5(RESET,LongRes,R,T,wd,wi,RAD,sigmaod)

%    matchpres = 1;                 %precisión del emparejamiento en numero de casillas(minimo
     1)

    Incremento = 0.1;                           %longitud de la casilla de posicion
        discretizada(metros)

    res = fix(LongRes / Incremento);    %dimension del mapa discreto en numero de celdas por
        lado

    r = R/6;                            %r(radio ruedas) R(radio Robot)

    persistent xpersistent;             %variables estáticas
    persistent ypersistent;
    persistent thetapersistent;
    persistent MAPpersistent;

    if RESET == 1                       %posición inicial
        xpersistent       = 0;
        ypersistent       = 0;
```

```
        thetapersistent   = 0;
        MAPpersistent     = zeros(res,res);
    else

        Vd = wd*r;                                %cinemática
        Vi = wi*r;
        V = (Vd+Vi)/2;
        W = (Vd-Vi)/((10*R)/6);

        thetapersistent = thetapersistent + W*T;
        xpersistent = xpersistent + V*cos(thetapersistent + pi/2)*T;
        ypersistent = ypersistent + V*sin(thetapersistent + pi/2)*T;

        %Calculo de dimensiones de la ventana de comparación
        deltatheta = (2 * sigmaod * r * T ) / (10/6 * R);                    %L = 10/6 * R
        deltax = (sigmaod * r * cos(thetapersistent + pi/2) * T) + ((deltatheta * T * r *(wd +
             wi) * sin(thetapersistent + pi/2))/2);
        deltay = (sigmaod * r * sin(thetapersistent + pi/2) * T) + ((deltatheta * T * r *(wd +
             wi) * cos(thetapersistent + pi/2))/2);

    end


    if RESET == 1
        %mapa de la primera iteración, 100% fiable
        MAPpersistent = CalculateMAP_v2(LongRes,Incremento,xpersistent,ypersistent,
            thetapersistent,RAD);
    else
        %vector "identidad" que indica la posición más favorable, AKA posición "ganadora"
        winner = zeros(1,27);
        w = 1;                                              %indice para el vector
        max = 0;                                  %valor de coincidencia máxima para comparar

        %con tres bucles concatenados, se prueban posiciones alrededor de las tres variables
            de la pose
        for i = -1:1
            xprobe = xpersistent + (deltax * i);
            for j = -1:1
                yprobe = ypersistent + (deltay * j);
                for k = -1:1
                    thetaprobe = thetapersistent + (deltatheta*k);
                    %se haya el mapa de la pose de prueba
                    MAPprobe = CalculateMAP_v2(LongRes,Incremento,xprobe,yprobe,thetaprobe,RAD
                        );

% Método de comparación 1(bucle^2)
%                     cuenta = 0;                         %ahora, se cuenta el numero de
    elementos en comun de el nuevo mapa con el mapa anterior
%                     for m = 1:matchpres:size(MAPprobe,1)
%                         for n = 1:matchpres:size(MAPprobe,2)
%                             if MAPprobe(m,n) == MAPpersistent(m,n)
%                                 cuenta = cuenta + 1;
%                             end
%                         end
%                     end
%                     if cuenta >= max          %si el recuento es mayor que el máximo visto
    hasta el momento, se marca esta posición como la nueva "ganadora"
%                         max = cuenta;
%                         winner = zeros(1,27);
%                         winner(w) = 1;
%                     end
%                     w = w + 1;

%Método de comparación 2(AND/mean2) mas eficiente, método anterior se deja comentado
                    double coincidencia;
                    coincidencia = mean2(MAPpersistent & MAPprobe);      %se calcula la
                        coincidencia de mapas

                    if coincidencia >= max      %si el recuento es mayor que el máximo visto
                        hasta el momento, se marca esta posición como la nueva "ganadora"
                        max = coincidencia;
                        winner = zeros(1,27);
                        winner(w) = 1;
```

```
                    end


                    w = w + 1;
                end
            end
        end
        w = 1;
        interrupt = 0;

        %tres bucles concatenados idénticos al anterior, para encontrar la posición "ganadora" y tomarla como válida
        for i = -1:1
            xprobe = xpersistent + (deltax * i);
            for j = -1:1
                yprobe = ypersistent + (deltay * j);
                for k = -1:1
                    thetaprobe = thetapersistent + (deltatheta*k);

                    %si es la ganadora, calcula el mapa, guarda la pose como válida y sale del bucle triple
                    if winner(w) == 1
                        newMAP = CalculateMAP_v2(LongRes,Incremento,xprobe,yprobe,thetaprobe, RAD);
                        xpersistent     = xprobe;
                        ypersistent     = yprobe;
                        thetapersistent = thetaprobe;
                        interrupt = 1;
                        break
                    end
                    w = w + 1;
                end
                if interrupt == 1
                    break
                end
            end
            if interrupt == 1
                break
            end
        end

        MAPpersistent = MAPpersistent | newMAP;  %se combina el mapa almacenado con los nuevos datos

    end


    %se vuelcan los resultados en las variables de salida. Matlab recomienda hacer esto cuando se trabaja con variables estáticas.
    x       = xpersistent;
    y       = ypersistent;
    theta   = thetapersistent;
    MAP     = MAPpersistent;

end
```

B.1.6. Mapping_v6

```
function [MAP,x,y,theta] = Mapping_v6(RESET,LongRes,R,T,wd,wi,RAD)
%Implementa la Occupancy Grid probabilística sin realizar map matching. Usa subfunción CalculateMAP_v3.

    Incremento = 0.1;                   %longitud de la casilla de posicion discretizada(metros)

    res = fix(LongRes / Incremento);    %longitud de la casilla de posicion discretizada(metros)

```

```
    r = R/6;                            %r(radio ruedas) R(radio Robot)

    persistent xpersistent;             %variables estáticas
    persistent ypersistent;
    persistent thetapersistent;
    persistent MAPpersistent;

    if RESET == 1                       %posición inicial
        xpersistent       = 0;
        ypersistent       = 0;
        thetapersistent   = 0;
        MAPpersistent     = 50*ones(res,res);

    else
        Vd = wd*r;                              %cinemática
        Vi = wi*r;
        V = (Vd+Vi)/2;
        W = (Vd-Vi)/((10*R)/6);

        thetapersistent = thetapersistent + W*T;
        xpersistent = xpersistent + V*cos(thetapersistent + pi/2)*T;
        ypersistent = ypersistent + V*sin(thetapersistent + pi/2)*T;
    end

    %calcula nueva Occupancy Grid
    MAPpersistent = CalculateMAP_v3(MAPpersistent,LongRes,Incremento,xpersistent,ypersistent,
        thetapersistent,RAD);

    MAPpersistent(MAPpersistent<0) = 0;         %fuerza valores dentro del rango elegido
    MAPpersistent(MAPpersistent>100)=100;
    MAP = (MAPpersistent/100);                  %escala valores antes de enviarlos fuera

    x       = xpersistent;                      %se vuelcan los resultados en las variables de
         salida. Matlab recomiendo hacer esto cuando se trabaja con variables estáticas.
    y       = ypersistent;
    theta   = thetapersistent;

end
```

B.1.7. Mapping_v7

```
function [MAP,x,y,theta] = Mapping_v7(RESET,LongRes,R,T,wd,wi,RAD,sigmaod)

    Incremento = 0.1;           %longitud de la casilla de posicion discretizada(metros)

    %dimension del mapa discreto en numero de celdas por lado
    res = fix(LongRes / Incremento);

    r = R/6;                    %r(radio ruedas) R(radio Robot)

    persistent xpersistent;                     %variables estáticas
    persistent ypersistent;
    persistent thetapersistent;
    persistent MAPpersistent;

    if RESET == 1                               %posición inicial
        xpersistent       = 0;
        ypersistent       = 0;
        thetapersistent   = 0;
        MAPpersistent     = 50*ones(res,res);
    else                                        %cinemática
        Vd = wd*r;
        Vi = wi*r;
        V = (Vd+Vi)/2;
        W = (Vd-Vi)/((10*R)/6);

        thetapersistent = thetapersistent + W*T;
        xpersistent = xpersistent + V*cos(thetapersistent + pi/2)*T;
```

```
28          ypersistent = ypersistent + V*sin(thetapersistent + pi/2)*T;
29
30          %Calculo de dimensiones de la ventana de comparación
31          deltatheta = (2 * sigmaod * r * T ) / (10/6 * R);                   %L = 10/6 * R
32          deltax = (sigmaod * r * cos(thetapersistent + pi/2) * T) + ((deltatheta * T * r *(wd +
                wi) * sin(thetapersistent + pi/2))/2);
33          deltay = (sigmaod * r * sin(thetapersistent + pi/2) * T) + ((deltatheta * T * r *(wd +
                wi) * cos(thetapersistent + pi/2))/2);
34
35      end
36
37          %mapa PROBABILISTICO de la primera iteración, 100% fiable
38      if RESET == 1
39          MAPpersistent = CalculateMAP_v3(MAPpersistent,LongRes,Incremento,xpersistent,
                ypersistent,thetapersistent,RAD);
40          MAPpersistent(MAPpersistent<0) = 0;
41          MAPpersistent(MAPpersistent>100)=100;
42
43      else
44
45          %vector "identidad" que indica la posición más favorable, AKA posición "ganadora"
46          winner = zeros(1,27);
47          w = 1;                                              %indice para el vector
48          max = 0;                                            %valor de coincidencia máxima para
                comparar
49
50          %con tres bucles concatenados, se prueban posiciones alrededor de las tres variables
                de la pose
51          for i = -1:1
52              xprobe = xpersistent + (deltax * i);
53              for j = -1:1
54                  yprobe = ypersistent + (deltay * j);
55                  for k = -1:1
56                      thetaprobe = thetapersistent + (deltatheta*k);
57                      %se haya el mapa BOOLEANO de la pose de prueba
58                      MAPprobe = CalculateMAP_v2(LongRes,Incremento,xprobe,yprobe,thetaprobe,RAD
                            );
59
60                      double coincidencia;
61
62                      %coincidencia como media del resultado de aplicar mapa de prueba como
                            máscara de mapa global
63                      coincidencia = mean(MAPpersistent(logical(MAPprobe)));
64
65                      %si el recuento es mayor que el máximo visto hasta el momento, se marca
                            esta posición como la nueva "ganadora"
66                      if coincidencia >= max
67                          max = coincidencia;
68                          winner = zeros(1,27);
69                          winner(w) = 1;
70                      end
71                      w = w + 1;
72                  end
73              end
74          end
75
76          w = 1;
77          interrupt = 0;
78
79          %tres bucles concatenados idénticos al anterior, para encontrar la posición "ganadora"
                y tomarla como válida
80          for i = -1:1
81              xprobe = xpersistent + (deltax * i);
82              for j = -1:1
83                  yprobe = ypersistent + (deltay * j);
84                  for k = -1:1
85                      thetaprobe = thetapersistent + (deltatheta*k);
86
87                      %si es la ganadora, actualiza el mapa global PROBABILISTICO, guarda la
                            pose como válida y sale del bucle triple
88                      if winner(w) == 1
89                          MAPpersistent = CalculateMAP_v3(MAPpersistent,LongRes,Incremento,
                                xprobe,yprobe,thetaprobe,RAD);
```

```
                        MAPpersistent(MAPpersistent<0) = 0;
                        MAPpersistent(MAPpersistent>100)=100;
                        xpersistent     = xprobe;
                        ypersistent     = yprobe;
                        thetapersistent = thetaprobe;
                        interrupt = 1;
                        break
                    end
                    w = w + 1;
                end
                if interrupt == 1
                    break
                end
            end
            if interrupt == 1
                break
            end
        end

    end


    %se vuelcan los resultados en las variables de salida. Matlab recomiendo hacer esto cuando
         se trabaja con variables estáticas.
    x       = xpersistent;
    y       = ypersistent;
    theta   = thetapersistent;
    MAP     = (MAPpersistent/100);

end
```

B.1.8. Mapping_v8

```
function [MAP,x,y,theta] = Mapping_v8(RESET,LongRes,R,T,wd,wi,RAD,sigmaod)

%descomentar lineas comentadas para debugging

    matchpres = 1;       %precisión del emparejamiento en numero de casillas(minimo 1)

    Incremento = 0.1;    %longitud de la casilla de posicion discretizada(metros)

    res = fix(LongRes / Incremento); %dimension del mapa discreto en numero de celdas por lado

    r = R/6;             %r(radio ruedas) R(radio Robot)

    persistent xpersistent;                     %variables estáticas
    persistent ypersistent;
    persistent thetapersistent;
    persistent MAPpersistent;

    %posición inicial, inicialización de variables estáticas
    if RESET == 1
        xpersistent       = 0;
        ypersistent       = 0;
        thetapersistent   = 0;
        MAPpersistent     = 50*ones(res,res);
    else
        Vd = wd*r;                              %cinemática
        Vi = wi*r;
        V = (Vd+Vi)/2;
        W = (Vd-Vi)/((10*R)/6);

        thetapersistent = thetapersistent + W*T;
        xpersistent = xpersistent + V*cos(thetapersistent + pi/2)*T;
        ypersistent = ypersistent + V*sin(thetapersistent + pi/2)*T;

        %Calculo de dimensiones de la ventana de comparación
        deltatheta = (2 * sigmaod * r * T ) / (10/6 * R);                   %L = 10/6 * R
```

```
36          deltax = (sigmaod * r * cos(thetapersistent + pi/2) * T) + ((deltatheta * T * r *(wd +
                wi) * sin(thetapersistent + pi/2))/2);
37          deltay = (sigmaod * r * sin(thetapersistent + pi/2) * T) + ((deltatheta * T * r *(wd +
                wi) * cos(thetapersistent + pi/2))/2);
38      end
39
40
41      %mapa probabilístico de la primera iteración, 100 % fiable
42      if RESET == 1
43          MAPpersistent = CalculateMAP_v3(MAPpersistent,LongRes,Incremento,xpersistent,
                ypersistent,thetapersistent,RAD);
44          MAPpersistent(MAPpersistent<0) = 0;
45          MAPpersistent(MAPpersistent>100)=100;
46      else
47          %vector "identidad" que indica la posición más favorable, AKA posición "ganadora"
48          winner = zeros(1,27);
49          w = 1;                          %indice para el vector
50          min = NaN;                      %para identificar primera medida
51
52          %con tres bucles concatenados, se prueban posiciones alrededor de las tres variables
                de la pose
53          for i = -1:1
54              xprobe = xpersistent + (deltax * i);
55              for j = -1:1
56                  yprobe = ypersistent + (deltay * j);
57                  for k = -1:1
58                      thetaprobe = thetapersistent + (deltatheta*k);
59
60                      %se calcula mapa de prueba a partir de mapa global
61                      MAPprobe = CalculateMAP_v3(MAPpersistent,LongRes,Incremento,xprobe,yprobe,
                            thetaprobe,RAD);
62                      MAPprobe(MAPprobe<0) = 0;
63                      MAPprobe(MAPprobe>100)=100;
64
65                      %se calcula el factor de correspondencia inversa con bucle doble
66                      correspondenciainv = 0;
67                      for m = 1:matchpres:size(MAPprobe,1)
68                          for n = 1:matchpres:size(MAPprobe,2)
69                              correspondenciainv = correspondenciainv + abs(MAPprobe(m,n) -
                                    MAPpersistent(m,n));
70                          end
71                      end
72
73  %                     correspondenciainv = correspondenciainv
74
75                      %si el recuento es menor que el mínimo visto hasta el momento o se detecta
                            primera medida, se marca esta posición como la nueva "ganadora"
76                      if (correspondenciainv <= min) || isnan(min)
77                          min = correspondenciainv;
78                          winner = zeros(1,27);
79                          winner(w) = 1;
80                      end
81                      w = w + 1;
82                  end
83              end
84          end
85
86  %         winner = winner
87
88          w = 1;
89          interrupt = 0;
90
91          %tres bucles concatenados idénticos al anterior, para encontrar la posición "ganadora"
                y tomarla como válida
92          for i = -1:1
93              xprobe = xpersistent + (deltax * i);
94              for j = -1:1
95                  yprobe = ypersistent + (deltay * j);
96                  for k = -1:1
97                      thetaprobe = thetapersistent + (deltatheta*k);
98
99                      %si es la ganadora, actualiza el mapa global probabilístico, guarda la
                            pose como válida y sale del bucle triple
```

```
                    if winner(w) == 1
                        MAPpersistent = CalculateMAP_v3(MAPpersistent,LongRes,Incremento,
                            xprobe,yprobe,thetaprobe,RAD);
                        MAPpersistent(MAPpersistent<0) = 0;
                        MAPpersistent(MAPpersistent>100)=100;
                        xpersistent     = xprobe;
                        ypersistent     = yprobe;
                        thetapersistent = thetaprobe;
                        interrupt = 1;
                        break
                    end
                    w = w + 1;
                end
                if interrupt == 1
                    break
                end
            end
            if interrupt == 1
                break
            end
        end


    end


    %se vuelcan los resultados en las variables de salida. Matlab recomiendo hacer esto cuando
         se trabaja con variables estáticas.
    x       = xpersistent;
    y       = ypersistent;
    theta   = thetapersistent;
    MAP     = (MAPpersistent/100);

end
```

B.1.9. Mapping_v9

```
function [MAP,x,y,theta] = Mapping_v9(RESET,LongRes,R,T,wd,wi,RAD,sigmaod)


    Incremento = 0.1;           %longitud de la casilla de posicion discretizada(metros)
    dramalevel = 5;             %parámetros de dramatizaciób y umbralización
    umbSup = 60;
    umbInf = 40;

    res = fix(LongRes / Incremento);    %dimension del mapa discreto en numero de celdas por
        lado

    r = R/6;                    %r(radio ruedas) R(radio Robot)

    persistent xpersistent;             %variables estáticas
    persistent ypersistent;
    persistent thetapersistent;
    persistent MAPpersistent;

    if RESET == 1                       %posición inicial
        xpersistent       = 0;
        ypersistent       = 0;
        thetapersistent   = 0;
        MAPpersistent     = 50*ones(res,res);
    else
        Vd = wd*r;                  %cinemática
        Vi = wi*r;
        V = (Vd+Vi)/2;
        W = (Vd-Vi)/((10*R)/6);

        thetapersistent = thetapersistent + W*T;
        xpersistent = xpersistent + V*cos(thetapersistent + pi/2)*T;
```

```
31          ypersistent = ypersistent + V*sin(thetapersistent + pi/2)*T;
32
33          %Calculo de dimensiones de la ventana de comparación
34          deltatheta = (2 * sigmaod * r * T ) / (10/6 * R);                   %L = 10/6 * R
35          deltax = (sigmaod * r * cos(thetapersistent + pi/2) * T) + ((deltatheta * T * r *(wd +
                wi) * sin(thetapersistent + pi/2))/2);
36          deltay = (sigmaod * r * sin(thetapersistent + pi/2) * T) + ((deltatheta * T * r *(wd +
                wi) * cos(thetapersistent + pi/2))/2);
37      end
38
39
40          %mapa de la primera iteración, 100% fiable
41      if RESET == 1
42          for i = 1:dramalevel     %bucle de dramatización
43              MAPpersistent = CalculateMAP_v3(MAPpersistent,LongRes,Incremento,xpersistent,
                    ypersistent,thetapersistent,RAD);
44          end
45          MAPpersistent(MAPpersistent<0) = 0;
46          MAPpersistent(MAPpersistent>100)=100;
47      else
48          %vector "identidad" que indica la posición más favorable, AKA posición "ganadora"
49          winner = zeros(1,27);
50          w = 1;                   %indice para el vector
51          max = 0;
52
53          %con tres bucles concatenados, se prueban posiciones alrededor de las tres variables
                de la pose
54          for i = -1:1
55              xprobe = xpersistent + (deltax * i);
56              for j = -1:1
57                  yprobe = ypersistent + (deltay * j);
58                  for k = -1:1
59                      thetaprobe = thetapersistent + (deltatheta*k);
60
61                      %calcula mapa BOOLEANO para la pose de prueba
62                      MAPprobe = CalculateMAP_v2(LongRes,Incremento,xprobe,yprobe,thetaprobe,RAD
                            );
63
64                      double coincidencia;
65
66                      %calcula coincidencia mediante umbralizacion del mapa gobal y comparacion
                            booleana
67                      coincidencia = mean2(((MAPprobe==1) & (MAPpersistent > umbSup)) | ((
                            MAPprobe==0) & (MAPpersistent < umbInf)));
68
69                      %si el recuento es mayor que el máximo visto hasta el momento o se detecta
                             primera medida, se marca esta posición como la nueva "ganadora"
70                      if  coincidencia >= max
71                          max = coincidencia;
72                          winner = zeros(1,27);
73                          winner(w) = 1;
74                      end
75
76                      w = w + 1;
77                  end
78              end
79          end
80
81
82          w = 1;
83          interrupt = 0;
84
85          %tres bucles concatenados idénticos al anterior, para encontrar la posición "ganadora"
                 y tomarla como válida
86          for i = -1:1
87              xprobe = xpersistent + (deltax * i);
88              for j = -1:1
89                  yprobe = ypersistent + (deltay * j);
90                  for k = -1:1
91                      thetaprobe = thetapersistent + (deltatheta*k);
92
93                      %si es la ganadora, actualiza el mapa global probabilístico, guarda la
                            pose como válida y sale del bucle triple
```

```
                    if winner(w) == 1
                        MAPpersistent = CalculateMAP_v3(MAPpersistent,LongRes,Incremento,
                            xprobe,yprobe,thetaprobe,RAD);
                        MAPpersistent(MAPpersistent<0) = 0;
                        MAPpersistent(MAPpersistent>100)=100;
                        xpersistent     = xprobe;
                        ypersistent     = yprobe;
                        thetapersistent = thetaprobe;
                        interrupt = 1;
                        break
                    end
                    w = w + 1;
                end
                if interrupt == 1
                    break
                end
            end
            if interrupt == 1
                break
            end
        end

    end

    %se vuelcan los resultados en las variables de salida. Matlab recomiendo hacer esto cuando
         se trabaja con variables estáticas.
    x       = xpersistent;
    y       = ypersistent;
    theta   = thetapersistent;
    MAP     = (MAPpersistent/100);

end
```

B.2. VissionSimulation

B.2.1. VissionSimulation_v1

```
function [RAD] = VissionSimulation_v1(OBSX,OBSY,DiagMax,x,y,theta)

    AngInicial = 225;
    Paso = 0.5;                   %valor positivo
    AngFinal = -45;


    NumMedidas = abs((AngInicial - AngFinal)/Paso) + 1;

    if AngInicial > AngFinal
        Paso = -Paso;
    end

    T01 = [ cos(theta)  -sin(theta)       x;
            sin(theta)   cos(theta)       y;
            0               0                   1 ];

    T10 = inv(T01);             %matriz de transformación inversa

    OBS = [ OBSX;
            OBSY;
            ones(size(OBSX))];


    OBSp = T10 * OBS;           %cambia obstaculos a SR del robot

    OBSpX = OBSp(1,:);
    OBSpY = OBSp(2,:);

```

```

    ANG = AngInicial : Paso : AngFinal;
    RAD = zeros(1,NumMedidas);

    for i = 1 : NumMedidas

        [X,Y] = polyxpoly(OBSpX,OBSpY,[0 DiagMax * cos(ANG(i)*pi/180)],[0 DiagMax * sin(ANG(i)
            *pi/180)]);    %encuentra intersecciones entre el haz de luz y los obstaculos

        if isempty(X)         %si no se encuentra intersección
            RAD(i) = NaN;

        else
            for j = 1 : length(X)  %se calcula la distancia a las intersecciones y se coge la
                menor
                DIS(j) = sqrt((X(j)^2)+(Y(j)^2));
            end

            RAD(i) = min(DIS);
            clear DIS;
        end
    end
end
```

B.3. MappingTB

B.3.1. MappingTB_v1

```
function [] = MappingTB_v1()

%Prueba el simulador con un mapa sencillo y una trayectoria circular, usa mapping_v1. Hace un
    hold para mostrar los resultados de cada iteración.

    CentroX = 10;         %caracteristicas de la trayectoria circular
    CentroY = 10;
    Radio = 9;

    LRecinto = 20;         %lado del recinto cuadrado

    OBS1XLimit = [5 8];    %posicion del obstaculo 1
    OBS1YLimit = [4 8];

    OBS2XLimit = [12 14]; %posicion del obstaculo 2
    OBS2YLimit = [12 14];

    OBS1X = OBS1XLimit([1 1 2 2 1]);
    OBS1Y = OBS1YLimit([1 2 2 1 1]);
    D
    OBS2X = OBS2XLimit([1 1 2 2 1]);
    OBS2Y = OBS2YLimit([1 2 2 1 1]);

    OBSX = [OBS1X NaN OBS2X];
    OBSY = [OBS1Y NaN OBS2Y];

    RecintoX = [0   0           LRecinto    LRecinto    0];
    RecintoY = [0   LRecinto    LRecinto    0           0];

    hold off;             %dibuja recinto y obstaculos
    plot(RecintoX, RecintoY, 'g');
    mapshow(OBSX,OBSY,'DisplayType','polygon','LineStyle','none');
    hold on;              %activa el 'hold'
    axis equal;



    for i = 0 : 0.1: 2*pi %trayectoria circular
```

```
38
39          x = CentroX + Radio*cos(i);
40          y = CentroY + Radio*sin(i);
41          [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,i);    %Simulador de
                telemetro
42          plot(x,y,'ob');        %dibuja robot
43          [MAPX,MAPY] = Mapping_v1(x,y,i,RAD);    %llama al mapper
44          plot(MAPX,MAPY,'xr');   %dibuja impactos
45          pause(0.01);
46      end
47
48      hold off;
49  end
```

B.3.2. MappingTB_v2

B.3.2.1. MappingTB_v2_1

```
1   function [] = MappingTB_v2_1()
2
3   %Entorno de simulación para mapping_v1. Mejoras respecto a la versiónanterior: Dibuja el robot
        mediante la función dibujar_v1; almacena los resultados en un solo vector de mapa; no
        hace hold
4
5       RadioRobot = 1;             %dimensiones del Robot
6
7       CentroX = 10;               %trayectoria circular
8       CentroY = 10;
9       Radio = 9;
10
11      LRecinto = 20;              %lado del Recinto cuadrado
12
13      OBS1XLimit = [5 8];         %Obstáculo 1
14      OBS1YLimit = [4 8];
15
16      OBS2XLimit = [12 14];       %Obstáculo 2
17      OBS2YLimit = [12 14];
18
19      OBS1X = OBS1XLimit([1 1 2 2 1]);
20      OBS1Y = OBS1YLimit([1 2 2 1 1]);
21
22      OBS2X = OBS2XLimit([1 1 2 2 1]);
23      OBS2Y = OBS2YLimit([1 2 2 1 1]);
24
25      OBSX = [OBS1X NaN OBS2X];
26      OBSY = [OBS1Y NaN OBS2Y];
27
28      RecintoX = [0   0           LRecinto    LRecinto    0];
29      RecintoY = [0   LRecinto    LRecinto    0           0];
30
31
32      for i = 0 : 0.1: 2*pi       %bucle de puntos de la trayectoria
33
34          x = CentroX + Radio*cos(i);        %Calcula punto
35          y = CentroY + Radio*sin(i);
36          [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,i);    %Simula el
                telemetro
37          [newmapX,newmapY] = Mapping_v1(x,y,i,RAD);                           %mapeo
38          if i == 0                                                            %almacena mapa
39              MAPX = newmapX;
40              MAPY = newmapY;
41          else
42              MAPX = [MAPX newmapX];
43              MAPY = [MAPY newmapY];
44          end
45          plot(RecintoX,RecintoY,'-.k',MAPX,MAPY,'.r');                        %dibuja
46          axis equal;
47          hold on;
```

```
48          mapshow(OBSX,OBSY,'DisplayType','polygon','LineStyle','none');
49          dibujar_v1(RadioRobot,x,y,i);
50          hold off;
51          pause(0.01);
52      end
53
54  end
```

B.3.2.2. MappingTB_v2_2

```
1   function [] = MappingTB_v2_2()
2
3   %Similar v2_1 pero con un mapa diferente.
4
5       RadioRobot = 1;         %dimensiones del Robot
6
7       CentroX = 10;           %trayectoria circular
8       CentroY = 10;
9       Radio = 9;
10
11      LRecinto = 20;          %lado del recinto cuadrado
12
13
14      %vectores de obstaculos
15      OBSX = [4 4 6 7 8 6 4 NaN 5  5  6  6  8  8  9  9  5  NaN 12 14 14 12 NaN 13 13 16 16 13
            NaN 12 12 16 16 12 NaN 8  8  11 11 8];
16      OBSY = [6 7 9 7 6 7 6 NaN 13 15 15 14 14 15 15 13 13 NaN 14 16 14 14 NaN 11 13 12 11 11
            NaN 7  8  8  7  7  NaN 9  12 12 9  9];
17
18      RecintoX = [0   0           LRecinto    LRecinto    0];
19      RecintoY = [0   LRecinto    LRecinto    0           0];
20
21
22      for i = 0 : 0.1: 2*pi       %bucle de puntos de la trayectoria
23
24          x = CentroX + Radio*cos(i);     %Calcula punto
25          y = CentroY + Radio*sin(i);
26          [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,i);    %Simula el
                telemetro
27          [newmapX,newmapY] = Mapping_v1(x,y,i,RAD);                          %mapeo
28          if i == 0                                                           %almacena mapa
29              MAPX = newmapX;
30              MAPY = newmapY;
31          else
32              MAPX = [MAPX newmapX];
33              MAPY = [MAPY newmapY];
34          end
35          plot(RecintoX,RecintoY,'-.k',MAPX,MAPY,'.r');                       %dibuja
36          axis equal;
37          hold on;
38          mapshow(OBSX,OBSY,'DisplayType','polygon','LineStyle','none');
39          dibujar_v1(RadioRobot,x,y,i);
40          hold off;
41          pause(0.01);
42      end
43
44  end
```

B.3.3. MappingTB_v3

```
function [] = MappingTB_v3_1()

    %Entorno de siumulación para mapping_v2, el cual recibbe como entrada el vector de
        velocidades instantaneas para poder hacer el calculo de posiciones.

    LRecinto = 20;       %lado del recinto cuadrado
    T = 0.1;             %periodo en segundos
    RadioRobot = 1;      %Dimensiones del Robot

    %Posición de los Obstaculos
    OBSX = [4 4 6 7 8 6 4 NaN 5  5  6  6  8  8  9  9  5  NaN 12 14 14 12 NaN 13 13 16 16 13
        NaN 12 12 16 16 12 NaN 8  8  11 11 8];
    OBSY = [6 7 9 7 6 7 6 NaN 13 15 15 14 14 15 15 13 13 NaN 14 16 14 14 NaN 11 13 12 11 11
        NaN 7  8  8  7  7  NaN 9  12 12 9  9];

   %velocidades angulares instantáneas
    Wd = [60*ones(1,13)  55*ones(1,10)  60*ones(1,10) 0*ones(1,8)  60*ones(1,15)   33*ones
        (1,7) 50*ones(1,20)];
    Wi = [60*ones(1,13)  72*ones(1,10)  60*ones(1,10) 18*ones(1,8)  60*ones(1,15)  -33*ones
        (1,7) 50*ones(1,20)];

    RecintoX = [0   0           LRecinto    LRecinto    0];
    RecintoY = [0   LRecinto    LRecinto    0           0];

    x = 0;                %inicialización de la pose
    y = 0;
    theta = 0;

    r = RadioRobot/6;     %radio de las ruedas

    Mapping_v2(1,RadioRobot,T,Wd(1),Wi(1),ones(541));         %primera llamara con Reset

    for i = 1:length(Wd)

        Vd = Wd(i)*r;     %cinemática
        Vi = Wi(i)*r;
        V = (Vd+Vi)/2;
        W = (Vd-Vi)/((10*RadioRobot)/6);

        theta = theta + W*T;
        x = x + V*cos(theta+pi/2)*T;
        y = y + V*sin(theta+pi/2)*T;


        [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,(theta));      %
            simulación de telemetro
        [newmapX,newmapY] = Mapping_v2(0,RadioRobot,T,Wd(i),Wi(i),RAD);              %mapeo

        if i == 1                                                       %almacena en mapa global
            MAPX = newmapX;
            MAPY = newmapY;
        else
            MAPX = [MAPX newmapX];
            MAPY = [MAPY newmapY];
        end
        plot(RecintoX,RecintoY,'-.k',MAPX,MAPY,'.r');                                %pinta
        axis equal;
        hold on;
        mapshow(OBSX,OBSY,'DisplayType','polygon','LineStyle','none');
        dibujar_v1(1,x,y,theta);
        hold off;
        pause(T);
    end

end
```

B.3.4. MappingTB_v4

```
function [] = MappingTB_v4_1(SigmaOd, SigmaMed) %Sigma(desviación típica) de Odometría y de Medida

%Entorno de simulación de mapping_v3, al que envía datos con error gaussiano, las desviaciones tipicas se introducen como variables de entrada. Dibuja posición real y posición calculada por mapping. Usa dibujar_v2. En la primera iteración, inicializar mapping_v3.
%descomentar lineas comentadas para generar tabla de error de pose y gráfica.

    LRecinto = 20;                                      %lado del recinto cuadrado
    T = 0.1;                                            %periodo en segundos
    RadioRobot = 1;                                     %Dimensiones del Robot

    %Posición de los Obstaculos
    OBSX = [4 4 6 7 8 6 4 NaN 5  5  6  6  8  8  9  9  5  NaN 12 14 14 12 NaN 13 13 16 16 13 NaN 12 12 16 16 12 NaN 8  8  11 11 8];
    OBSY = [6 7 9 7 6 7 6 NaN 13 15 15 14 14 15 15 13 13 NaN 14 16 14 14 NaN 11 13 12 11 11 NaN 7  8  8  7  7  NaN 9  12 12 9  9];

    %velocidades angulares instantáneas
    Wd = [60*ones(1,13)  55*ones(1,10)  60*ones(1,10) 0*ones(1,8)  60*ones(1,15)   33*ones(1,7) 50*ones(1,20)];
    Wi = [60*ones(1,13)  72*ones(1,10)  60*ones(1,10) 18*ones(1,8)  60*ones(1,15)  -33*ones(1,7) 50*ones(1,20)];

    RecintoX = [0   0           LRecinto    LRecinto    0];
    RecintoY = [0   LRecinto    LRecinto    0           0];

    x = 0;                                              %inicialización de la pose
    y = 0;
    theta = 0;

    r = RadioRobot/6;                                   %radio de las ruedas

    %primera iteración en posición inicial y sin error
    [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);
    [newmapX,newmapY,xp,yp,thetap] = Mapping_v3(1,RadioRobot,T,0,0,RAD);

    MAPX = newmapX;                                     %almacena en mapa global
    MAPY = newmapY;

    dibujar_v2(RadioRobot,x,y,theta,'g');                       %dibuja pose real y calculada( deben ser iguales)
    axis equal;
    hold on;
    mapshow(OBSX,OBSY,'DisplayType','polygon','LineStyle','none');
    plot(RecintoX,RecintoY,'-.k',MAPX,MAPY,'.r');
    dibujar_v2(RadioRobot,xp,yp,thetap,'k');
    hold off;

    pause(0.1);

    Wdp = Wd + SigmaOd*randn(size(Wd));                 %incluye error en velocidades angulares
    Wip = Wi + SigmaOd*randn(size(Wi));

%     iteracion = linspace(1,length(Wd)+1,length(Wd)+1)';
%     poseX = zeros(length(Wd)+1,1);
%     poseY = zeros(length(Wd)+1,1);
%     poseTHETA = zeros(length(Wd)+1,1);
%     error = zeros(length(Wd)+1,1);

%     poseX(1) = x;
%     poseY(1) = y;
%     poseTHETA(1) = theta;
%     error(1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);


    for i = 1:length(Wd)                                %bucle de instantes de tiempo discreto

        Vd = Wd(i)*r;                                   %cinemática
        Vi = Wi(i)*r;
```

```
        V = (Vd+Vi)/2;
        W = (Vd-Vi)/((10*RadioRobot)/6);

        theta = theta + W*T;
        x = x + V*cos(theta+pi/2)*T;
        y = y + V*sin(theta+pi/2)*T;


        [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta); %simulador telémetro

        RAD = RAD + SigmaMed*randn(size(RAD));        %incluye error en impactos de telémetro

        [newmapX,newmapY,xp,yp,thetap] = Mapping_v3(0,RadioRobot,T,Wdp(i),Wip(i),RAD);      % mapeo

        MAPX = [MAPX newmapX];                                              %almacena en mapa global
        MAPY = [MAPY newmapY];

        dibujar_v2(RadioRobot,x,y,theta,'g');                   %dibuja pose real en verde
        axis equal;
        hold on;
        mapshow(OBSX,OBSY,'DisplayType','polygon','LineStyle','none');   %dibuja entorno
        plot(RecintoX,RecintoY,'-.k',MAPX,MAPY,'.r');                     %dibuja mapa
        dibujar_v2(RadioRobot,xp,yp,thetap,'k');                %dibuja pose calculada en negro
        hold off;
        pause(0.01);

%         poseX(i+1) = x;
%         poseY(i+1) = y;
%         poseTHETA(i+1) = theta;
%         error(i+1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);
    end

%    print_table([iteracion round(poseX,3) round(poseY,3) round(poseTHETA,3) round(error,4)],{' %.4g'},'printMode','latex','printLatexFull',false)
%        errorcuadraticomedio = mean(error)
%    figure(3)
%    plot(error,'-s')
%    xlabel('Iteración')
%    ylabel('Error cuadrático')
%    grid on


end
```

B.3.5. MappingTB_v5

B.3.5.1. MappingTB_v5_1

```
function [] = MappingTB_v5_1(SigmaOd, SigmaMed) %Sigma(desviación típica) de Odometría y de Medida

%Entorno de simulación para mapping_v4, que trabaja con un mapa de posiciones discretizadas (occupancy grid). Por problemas de dibujo, se cambia la técnica de dibujar los obstáculos. No almacena los resultados de las iteraciones, ya que mapping_v4 envía el mapa completo en cada iteración.. En la primera iteración, inicializar mapping_v4.
    LRecinto = 20;                                  %lado del recinto cuadrado
    T = 0.1;                                        %periodo en segundos
    RadioRobot = 1;                                 %Dimensiones del Robot

    %Posición de los Obstaculos
    OBSX = [4 4 6 7 8 6 4 NaN 5  5  6  6  8  8  9  9  5  NaN 12 14 14 12 NaN 13 13 16 16 13 NaN 12 12 16 16 12 NaN 8  8  11 11 8];
    OBSY = [6 7 9 7 6 7 6 NaN 13 15 15 14 14 15 15 13 13 NaN 14 16 14 14 NaN 11 13 12 11 11 NaN 7  8  8  7  7  NaN 9  12 12 9  9];
```

```
11
12      %velocidades angulares instantáneas
13      Wd = [60*ones(1,13)  55*ones(1,10)  60*ones(1,10) 0*ones(1,8)  60*ones(1,15)   33*ones
            (1,7) 50*ones(1,20)];
14      Wi = [60*ones(1,13)  72*ones(1,10)  60*ones(1,10) 18*ones(1,8)  60*ones(1,15)  -33*ones
            (1,7) 50*ones(1,20)];
15
16      RecintoX = [0   0           LRecinto    LRecinto    0];
17      RecintoY = [0   LRecinto    LRecinto    0           0];
18
19      x = 0;                                                          %inicialización de la
            pose
20      y = 0;
21      theta = 0;
22
23      r = RadioRobot/6;                               %radio de las ruedas
24
25      %primera iteración en posición inicial y sin error
26      [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);
27      [MAP,xp,yp,thetap] = Mapping_v4(1,LRecinto,RadioRobot,T,0,0,RAD);
28
29      %estructura de parámetros para mapshow
30      R = maprasterref('XLimWORLD',[0 LRecinto],'YLimWorld',[0 LRecinto],'RasterSize',size(MAP))
            ;
31
32      mapshow(not(MAP),R);                            %dibuja mapa
33      axis equal;
34      hold on;
35      plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');    %dibuja recinto y obstáculos
36      dibujar_v2(RadioRobot,x,y,theta,'y');           %dibuja pose real en amarillo
37      dibujar_v2(RadioRobot,xp,yp,thetap,'k');        %dibuja pose calculada en negro
38      hold off;
39      pause(0.1);
40
41      Wdp = Wd + SigmaOd*randn(size(Wd));             %incluye error en velocidades angulares
42      Wip = Wi + SigmaOd*randn(size(Wi));
43
44      for i = 1:length(Wd)                            %bucle de instantes de tiempo discreto
45
46          Vd = Wd(i)*r;                               %cinemática
47          Vi = Wi(i)*r;
48          V = (Vd+Vi)/2;
49          W = (Vd-Vi)/((10*RadioRobot)/6);
50
51          theta = theta + W*T;
52          x = x + V*cos(theta+pi/2)*T;
53          y = y + V*sin(theta+pi/2)*T;
54
55          [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);  %simulador de
                telémetro
56
57          RAD = RAD + SigmaMed*randn(size(RAD));  %incluye error en impactos de telémetro
58
59          [MAP,xp,yp,thetap] = Mapping_v4(0,LRecinto,RadioRobot,T,Wdp(i),Wip(i),RAD); %mapeo
60
61
62          mapshow(not(MAP),R);                        %dibuja mapa
63          axis equal;
64          hold on;
65          plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');  %dibuja recinto y obstáculos
66          dibujar_v2(RadioRobot,x,y,theta,'y');       %dibuja pose real en amarillo
67          dibujar_v2(RadioRobot,xp,yp,thetap,'k');    %dibuja pose calculada en negro
68          hold off;
69          pause(0.1);
70
71      end
72
73      DisplayMap(MAP,2);                  %muestra el mapa en figura 2
74  end
```

B.3.5.2. MappingTB_v5_2

```
function [] = MappingTB_v5_2(SigmaOd, SigmaMed) %Sigma(desviación típica) de Odometría y de
    Medida

%Similar a v5_1, pero con otro mapa y distinta base de tiempos.

    LRecinto = 20;                              %lado del recinto cuadrado
    T = 0.5;                                    %periodo en segundos
    RadioRobot = 1;                             %Dimensiones del Robot

    %Posición de los Obstaculos
    OBSX = [1  3  9  1  NaN 13 13 14 14 15 15 16 16 13 NaN 16 19 19 16 NaN 10 11 11 12 12 14
        14 10 10];
    OBSY = [12 10 18 12 NaN 19 18 18 17 17 16 16 19 19 NaN 12 14 10 12 NaN 7  7  4  4  7  7  3
         3  7 ];

    %velocidades angulares instantáneas
    Wd = [12*ones(1,5)  11*ones(1,14)   -7*ones(1,5)   12*ones(1,14)   5.9*ones(1,8) 10*ones
        (1,20)];
    Wi = [15*ones(1,5)  11*ones(1,14)    0*ones(1,5)   12*ones(1,14)  -5.9*ones(1,8) 11.5*ones
        (1,20)];

    RecintoX = [0   0           LRecinto    LRecinto    0];
    RecintoY = [0   LRecinto    LRecinto    0           0];

    x = 0;                                      %inicialización de la pose
    y = 0;
    theta = 0;

    r = RadioRobot/6;                           %radio de las ruedas

    %primera iteración en posición inicial y sin error
    [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);
    [MAP,xp,yp,thetap] = Mapping_v4(1,LRecinto,RadioRobot,T,0,0,RAD);

    %estructura de parámetros para mapshow
    R = maprasterref('XLimWORLD',[0 LRecinto],'YLimWorld',[0 LRecinto],'RasterSize',size(MAP))
        ;

    mapshow(not(MAP),R);                                %dibuja mapa
    axis equal;
    hold on;
    plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');  %dibuja recinto y obstáculos
    dibujar_v2(RadioRobot,x,y,theta,'y');         %dibuja pose real en amarillo
    dibujar_v2(RadioRobot,xp,yp,thetap,'k');      %dibuja pose calculada en negro
    hold off;
    pause(0.1);

    Wdp = Wd + SigmaOd*randn(size(Wd));           %incluye error en velocidades angulares
    Wip = Wi + SigmaOd*randn(size(Wi));

    for i = 1:length(Wd)                          %bucle de instantes de tiempo discreto

        Vd = Wd(i)*r;                             %cinemática
        Vi = Wi(i)*r;
        V = (Vd+Vi)/2;
        W = (Vd-Vi)/((10*RadioRobot)/6);

        theta = theta + W*T;
        x = x + V*cos(theta+pi/2)*T;
        y = y + V*sin(theta+pi/2)*T;

        [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);  %simulador de
            telémetro

        RAD = RAD + SigmaMed*randn(size(RAD));    %incluye error en impactos de telémetro

        [MAP,xp,yp,thetap] = Mapping_v4(0,LRecinto,RadioRobot,T,Wdp(i),Wip(i),RAD); %mapeo


        mapshow(not(MAP),R);                            %dibuja mapa
```

```
64          axis equal;
65          hold on;
66          plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');   %dibuja recinto y obstáculos
67          dibujar_v2(RadioRobot,x,y,theta,'y');          %dibuja pose real en amarillo
68          dibujar_v2(RadioRobot,xp,yp,thetap,'k');       %dibuja pose calculada en negro
69          hold off;
70          pause(0.1);
71
72      end
73
74      DisplayMap(MAP,2);                    %muestra el mapa en figura 2
75
76  end
```

B.3.6. MappingTB_v6

B.3.6.1. MappingTB_v6_1

```
1   function [] = MappingTB_v6_1(SigmaOd, SigmaMed)        %Sigma(desviación típica) de Odometría y
        de Medida
2
3   %Entorno de simulación para mapping_v5. Sin cambios de simulación.
4   %descomentar lineas comentadas para generar tabla de error de pose y gráfica.
5
6       LRecinto = 20;                                  %lado del recinto cuadrado
7       T = 0.1;                                        %periodo en segundos
8       RadioRobot = 1;                                 %Dimensiones del Robot
9
10      %Posición de los Obstaculos
11      OBSX = [4 4 6 7 8 6 4 NaN 5  5  6  6  8  8  9  9  5  NaN 12 14 14 12 NaN 13 13 16 16 13
            NaN 12 12 16 16 12 NaN 8  8  11 11 8];
12      OBSY = [6 7 9 7 6 7 6 NaN 13 15 15 14 14 15 15 13 13 NaN 14 16 14 14 NaN 11 13 12 11 11
            NaN 7  8  8  7  7  NaN 9  12 12 9  9];
13
14      %velocidades angulares instantáneas
15      Wd = [60*ones(1,13)  55*ones(1,10)  60*ones(1,10) 0*ones(1,8)  60*ones(1,15)   33*ones
            (1,7) 50*ones(1,20)];
16      Wi = [60*ones(1,13)  72*ones(1,10)  60*ones(1,10) 18*ones(1,8)  60*ones(1,15)  -33*ones
            (1,7) 50*ones(1,20)];
17
18      RecintoX = [0   0           LRecinto    LRecinto    0];
19      RecintoY = [0   LRecinto    LRecinto    0           0];
20
21      x = 0;                                           %inicialización de la pose
22      y = 0;
23      theta = 0;
24
25      r = RadioRobot/6;                               %radio de las ruedas
26
27      %primera iteración en posición inicial y sin error
28      [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);
29      [MAP,xp,yp,thetap] = Mapping_v5(1,LRecinto,RadioRobot,T,0,0,RAD,SigmaOd);
30
31      %estructura de parámetros para mapshowya
32
33      R = maprasterref('XLimWORLD',[0 LRecinto],'YLimWorld',[0 LRecinto],'RasterSize',size(MAP))
            ;
34
35      mapshow(not(MAP),R);                            %dibuja mapa
36      axis equal;
37      hold on;
38      plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');        %dibuja recinto y obstáculos
39      dibujar_v2(RadioRobot,x,y,theta,'g');                       %dibuja pose real en verde
40      dibujar_v2(RadioRobot,xp,yp,thetap,'k');            %dibuja pose calculada en negro
41      hold off;
42      pause(0.1);
43
```

```
44      Wdp = Wd + SigmaOd*randn(size(Wd));                    %incluye error en velocidades
            angulares
45      Wip = Wi + SigmaOd*randn(size(Wi));
46
47  %     iteracion = linspace(1,length(Wd)+1,length(Wd)+1)';
48  %     poseX = zeros(length(Wd)+1,1);
49  %     poseY = zeros(length(Wd)+1,1);
50  %     poseTHETA = zeros(length(Wd)+1,1);
51  %     error = zeros(length(Wd)+1,1);
52  %
53  %     poseX(1) = x;
54  %     poseY(1) = y;
55  %     poseTHETA(1) = theta;
56  %     error(1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);
57
58      for i = 1:length(Wd)                  %bucle de instantes de tiempo discreto
59
60          Vd = Wd(i)*r;                     %cinemática
61          Vi = Wi(i)*r;
62          V = (Vd+Vi)/2;
63          W = (Vd-Vi)/((10*RadioRobot)/6);
64
65          theta = theta + W*T;
66          x = x + V*cos(theta+pi/2)*T;
67          y = y + V*sin(theta+pi/2)*T;
68
69          [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);   %simulador de
                telémetro
70
71          RAD = RAD + SigmaMed*randn(size(RAD));      %incluye error en impactos de telémetro
72
73          [MAP,xp,yp,thetap] = Mapping_v5(0,LRecinto,RadioRobot,T,Wdp(i),Wip(i),RAD,SigmaOd); %
                mapeo
74
75
76          mapshow(not(MAP),R);                                          %dibuja mapa
77          axis equal;
78          hold on;
79          plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');            %dibuja recinto y obstáculos
80          dibujar_v2(RadioRobot,x,y,theta,'g');                   %dibuja pose real en verde
81          dibujar_v2(RadioRobot,xp,yp,thetap,'k');                %dibuja pose calculada en
                negro
82          hold off;
83          pause(0.1);
84
85  %         poseX(i+1) = x;
86  %         poseY(i+1) = y;
87  %         poseTHETA(i+1) = theta;
88  %         error(i+1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);
89
90      end
91
92  %     print_table([iteracion round(poseX,3) round(poseY,3) round(poseTHETA,3) round(error,4)
        ],{' %.4g'},'printMode','latex','printLatexFull',false)
93  %         errorcuadraticomedio = mean(error)
94  %     figure(3)
95  %     plot(error,'-s')
96  %     xlabel('Iteración')
97  %     ylabel('Error cuadrático')
98  %     grid on
99      DisplayMap(MAP,2);                                            %muestra el mapa en figura 2
100
101 end
```

B.3.6.2. MappingTB_v6_2

```
function [] = MappingTB_v6_2(SigmaOd, SigmaMed) %Sigma(desviación típica) de Odometría y de
    Medida

%Similar a v6_1, pero con otro mapa y distinta base de tiempos.
%descomentar lineas comentadas para generar tabla de error de pose y gráfica.

    LRecinto = 20;                              %lado del recinto cuadrado
    T = 0.5;                                    %periodo en segundos
    RadioRobot = 1;                             %Dimensiones del Robot

    %Posición de los Obstáculos
    OBSX = [1  3  9  1  NaN 13 13 14 14 15 15 16 16 13 NaN 16 19 19 16 NaN 10 11 11 12 12 14
        14 10 10];
    OBSY = [12 10 18 12 NaN 19 18 18 17 17 16 16 19 19 NaN 12 14 10 12 NaN 7  7  4  4  7  7  3
          3  7 ];

    %velocidades angulares instantáneas
    Wd = [12*ones(1,5)  11*ones(1,14)   -7*ones(1,5)   12*ones(1,14)   5.9*ones(1,8) 10*ones
        (1,20)];
    Wi = [15*ones(1,5)  11*ones(1,14)    0*ones(1,5)   12*ones(1,14)  -5.9*ones(1,8) 11.5*ones
        (1,20)];

    RecintoX = [0   0           LRecinto    LRecinto    0];
    RecintoY = [0   LRecinto    LRecinto    0           0];

    x = 0;                                                  %inicialización de la pose
    y = 0;
    theta = 0;

    r = RadioRobot/6;                           %radio de las ruedas

    %primera iteración en posición inicial y sin error
    [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);
    [MAP,xp,yp,thetap] = Mapping_v5(1,LRecinto,RadioRobot,T,0,0,RAD,SigmaOd);

    %estructura de parámetros para mapshow
    R = maprasterref('XLimWORLD',[0 LRecinto],'YLimWorld',[0 LRecinto],'RasterSize',size(MAP))
        ;

    mapshow(not(MAP),R);                                    %dibuja mapa
    axis equal;
    hold on;
    plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');        %dibuja recinto y obstáculos
    dibujar_v2(RadioRobot,x,y,theta,'g');               %dibuja pose real en verde
    dibujar_v2(RadioRobot,xp,yp,thetap,'k');            %dibuja pose calculada en negro
    hold off;
    pause(0.1);

    Wdp = Wd + SigmaOd*randn(size(Wd));         %incluye error en velocidades angulares
    Wip = Wi + SigmaOd*randn(size(Wi));

%     iteracion = linspace(1,length(Wd)+1,length(Wd)+1)';
%     poseX = zeros(length(Wd)+1,1);
%     poseY = zeros(length(Wd)+1,1);
%     poseTHETA = zeros(length(Wd)+1,1);
%     error = zeros(length(Wd)+1,1);
%
%     poseX(1) = x;
%     poseY(1) = y;
%     poseTHETA(1) = theta;
%     error(1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);

    for i = 1:length(Wd)                        %bucle de instantes de tiempo discreto

        Vd = Wd(i)*r;                           %cinemática
        Vi = Wi(i)*r;
        V = (Vd+Vi)/2;
        W = (Vd-Vi)/((10*RadioRobot)/6);

        theta = theta + W*T;
```

```
65          x = x + V*cos(theta+pi/2)*T;
66          y = y + V*sin(theta+pi/2)*T;
67
68          [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);    %simulador de
                telémetro
69
70          RAD = RAD + SigmaMed*randn(size(RAD));          %incluye error en impactos de
                telémetro
71
72          [MAP,xp,yp,thetap] = Mapping_v5(0,LRecinto,RadioRobot,T,Wdp(i),Wip(i),RAD,SigmaOd); %
                mapeo
73
74
75          mapshow(not(MAP),R);                            %dibuja mapa
76          axis equal;
77          hold on;
78          plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');    %dibuja recinto y obstáculos
79          dibujar_v2(RadioRobot,x,y,theta,'g');           %dibuja pose real en verde
80          dibujar_v2(RadioRobot,xp,yp,thetap,'k');        %dibuja pose calculada en negro
81          hold off;
82          pause(0.1);
83
84  %         poseX(i+1) = x;
85  %         poseY(i+1) = y;
86  %         poseTHETA(i+1) = theta;
87  %         error(i+1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);
88
89      end
90
91  %     print_table([iteracion round(poseX,3) round(poseY,3) round(poseTHETA,3) round(error,4)
        ],{' %.4g'},'printMode','latex','printLatexFull',false)
92  %         errorcuadraticomedio = mean(error)
93  %     figure(3)
94  %     plot(error,'-s')
95  %     xlabel('Iteración')
96  %     ylabel('Error cuadrático')
97  %     grid on
98      DisplayMap(MAP,2);                  %muestra el mapa en figura 2
99
100 end
```

B.3.6.3. MappingTB_v6_3

```
1   function [] = MappingTB_v6_3(SigmaOd, SigmaMed)          %Sigma(desviación típica) de Odometría
        y de Medida
2
3       %Entorno de simulación con un nuevo mapa de obstáculos que además, implementa obstáculos
            moviles con el fin de probar la capacidad de olvido frente a obstáculos no fijos en el
            tiempo.
4
5       LRecinto = 20;                                      %lado del recinto cuadrado
6       T = 0.5;                                            %periodo en segundos
7       RadioRobot = 1;                                     %Dimensiones del Robot
8
9       %Posición de los Obstaculos fijos
10      OBSX = [4   6   6   4   4    NaN    10  14  14  10    10  NaN    9   11  11  9   9
            NaN    2   4   4   2   2   NaN    17  19  19  17  17];
11      OBSY = [4   4   2   2   4    NaN    6   6   4   4     6   NaN    13  13  10  10  13
            NaN    16  16  14  14  16  NaN    12  12  8   8   12];
12
13      %velocidades angulares instantáneas
14      Wd = [12*ones(1,9)  8*ones(1,10)   19.3*ones(1,7)   12*ones(1,9)   -5.2*ones(1,6)  10*ones
            (1,15)   0*ones(1,15)    10*ones(1,18)  ];
15      Wi = [15*ones(1,9)  8*ones(1,10)   12*ones(1,7)    12*ones(1,9)    5.2*ones(1,6)  10*ones
            (1,15)  -6*ones(1,15)    11.2*ones(1,18)];
16
17      RecintoX = [0   0          LRecinto    LRecinto    0];
18      RecintoY = [0   LRecinto   LRecinto    0           0];
19
```

```
20      x = 0;                                              %inicialización de la pose
21      y = 0;
22      theta = 0;
23
24      r = RadioRobot/6;                                   %radio de las ruedas
25
26      %estado inicial de los obstáculos móviles
27      OBSXnew = [OBSX    NaN    14  15    15    14    14    NaN    7   8   8   7   7  NaN   9
                10   10   9   9];
28      OBSYnew = [OBSY    NaN    19  19    18    18    19    NaN    12  12  11  11  12 NaN   2
                2    1    1   2];
29
30      %primera iteración en posición inicial y sin error
31      [RAD] = VissionSimulation_v1(OBSXnew,OBSYnew,LRecinto * sqrt(2),x,y,theta);
32      [MAP,xp,yp,thetap] = Mapping_v5(1,LRecinto,RadioRobot,T,0,0,RAD,SigmaOd);
33
34      %estructura de parámetros para mapshow
35      R = maprasterref('XLimWORLD',[0 LRecinto],'YLimWorld',[0 LRecinto],'RasterSize',size(MAP))
            ;
36
37      mapshow(not(MAP),R);                                        %dibuja mapa
38      axis equal;
39      hold on;
40      plot(RecintoX,RecintoY,'-.k',OBSXnew,OBSYnew,'r');  %dibuja recinto y obstáculos
41      dibujar_v2(RadioRobot,x,y,theta,'g');               %dibuja pose real en verde
42      dibujar_v2(RadioRobot,xp,yp,thetap,'k');            %dibuja pose calculada en negro
43      hold off;
44      pause(0.1);
45
46      Wdp = Wd + SigmaOd*randn(size(Wd));         %incluye error en velocidades angulares
47      Wip = Wi + SigmaOd*randn(size(Wi));
48
49      for i = 1:length(Wd)                        %bucle de instantes de tiempo discreto
50
51          %actualiza la posición de los obstáculos móviles
52          OBSXnew = [OBSX    NaN    14+5*sin(i/10)    15+5*sin(i/10)    15+5*sin(i/10)    14+5*
                sin(i/10)    14+5*sin(i/10)    NaN    7-(i/5)   8-(i/5)   8-(i/5)   7-(i/5)   7-(
                i/5)  NaN   9+sin(i/5)    10+sin(i/5)    10+sin(i/5)    9+sin(i/5)  9+sin(i/5)
                ];
53          OBSYnew = [OBSY    NaN    19                  19                  18
                18                19          NaN    12          12        11        11
                12    NaN     2                 2              1               1          2
                ];
54
55          %elimina el objeto que se mueva mas alla de -1.5. Funciona solo para objetos de 4
                vertices
56          fuera = find(OBSXnew <= -1.5);
57          if not(isempty(fuera))
58              OBSXnew(fuera(1):(fuera(1)+5)) = NaN;
59          end
60
61          Vd = Wd(i)*r;                           %cinemática
62          Vi = Wi(i)*r;
63          V = (Vd+Vi)/2;
64          W = (Vd-Vi)/((10*RadioRobot)/6);
65
66          theta = theta + W*T;
67          x = x + V*cos(theta+pi/2)*T;
68          y = y + V*sin(theta+pi/2)*T;
69
70          %simulador de telémetro
71          [RAD] = VissionSimulation_v1(OBSXnew,OBSYnew,LRecinto * sqrt(2),x,y,theta);
72
73          %incluye error en impactos de telémetro
74          RAD = RAD + SigmaMed*randn(size(RAD));
75
76          %mapeo
77          [MAP,xp,yp,thetap] = Mapping_v5(0,LRecinto,RadioRobot,T,Wdp(i),Wip(i),RAD,SigmaOd);
78
79
80          mapshow(not(MAP),R);                                    %dibuja mapa
81          axis equal;
82          hold on;
```

```
        plot(RecintoX,RecintoY,'-.k',OBSXnew,OBSYnew,'r'); %dibuja recinto y obstáculos
        dibujar_v2(RadioRobot,x,y,theta,'g');              %dibuja pose real en verde
        dibujar_v2(RadioRobot,xp,yp,thetap,'k');       %dibuja pose calculada en negro
        hold off;
        pause(0.1);

    end

    DisplayMap(MAP,2);                  %muestra el mapa en figura 2

end
```

B.3.7. MappingTB_v7

B.3.7.1. MappingTB_v7_1

```
function [] = MappingTB_v7_1(SigmaOd, SigmaMed) %Sigma(desviación típica) de Odometría y de
    Medida

%Entorno de simulación para mapping_v6. Sin cambios de simulación.

    LRecinto = 20;              %lado del recinto cuadrado
    T = 0.1;                    %periodo en segundos
    RadioRobot = 1;             %Dimensiones del Robot

    %Posición de los Obstaculos
    OBSX = [4 4 6 7 8 6 4 NaN 5  5  6  6  8  8  9  9  5  NaN 12 14 14 12 NaN 13 13 16 16 13
        NaN 12 12 16 16 12 NaN 8  8  11 11 8];
    OBSY = [6 7 9 7 6 7 6 NaN 13 15 15 14 14 15 15 13 13 NaN 14 16 14 14 NaN 11 13 12 11 11
        NaN 7  8  8  7  7  NaN 9  12 12 9  9];

    %velocidades angulares instantáneas
    Wd = [60*ones(1,13)  55*ones(1,10)  60*ones(1,10) 0*ones(1,8)  60*ones(1,15)   33*ones
        (1,7) 50*ones(1,20)];
    Wi = [60*ones(1,13)  72*ones(1,10)  60*ones(1,10) 18*ones(1,8)  60*ones(1,15)  -33*ones
        (1,7) 50*ones(1,20)];

    RecintoX = [0   0           LRecinto    LRecinto    0];
    RecintoY = [0   LRecinto    LRecinto    0           0];

    x = 0;                      %inicialización de la pose
    y = 0;
    theta = 0;

    r = RadioRobot/6;           %radio de las ruedas

    %primera iteración en posición inicial y sin error
    [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);
    [MAP,xp,yp,thetap] = Mapping_v6(1,LRecinto,RadioRobot,T,0,0,RAD);

    %estructura de parámetros para mapshow
    R = maprasterref('XLimWORLD',[0 LRecinto],'YLimWorld',[0 LRecinto],'RasterSize',size(MAP))
        ;

    mapshow((1-MAP),R);                                 %dibuja mapa
    axis equal;
    hold on;
    plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');        %dibuja recinto y obstáculos
    dibujar_v2(RadioRobot,x,y,theta,'g');               %dibuja pose real en verde
    dibujar_v2(RadioRobot,xp,yp,thetap,'k');            %dibuja pose calculada en negro
    hold off;
    pause(0.1);

    Wdp = Wd + SigmaOd*randn(size(Wd));         %incluye error en velocidades angulares
    Wip = Wi + SigmaOd*randn(size(Wi));

    for i = 1:length(Wd)                        %bucle de instantes de tiempo discreto

```

```
        Vd = Wd(i)*r;                                %cinemática
        Vi = Wi(i)*r;
        V = (Vd+Vi)/2;
        W = (Vd-Vi)/((10*RadioRobot)/6);

        theta = theta + W*T;
        x = x + V*cos(theta+pi/2)*T;
        y = y + V*sin(theta+pi/2)*T;

        %simulador de telémetro
        [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);

        %incluye error en impactos de telémetro
        RAD = RAD + SigmaMed*randn(size(RAD));

        %mapeo
        [MAP,xp,yp,thetap] = Mapping_v6(0,LRecinto,RadioRobot,T,Wdp(i),Wip(i),RAD);

        mapshow((1-MAP),R);                          %invierte valores y dibuja mapa
        axis equal;
        hold on;
        plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');     %dibuja recinto y obstáculos
        dibujar_v2(RadioRobot,x,y,theta,'g');            %dibuja pose real en verde
        dibujar_v2(RadioRobot,xp,yp,thetap,'k');         %dibuja pose calculada en negro
        hold off;
        pause(0.1);

    end

    DisplayMap(MAP,2);                    %muestra el mapa en figura 2
end
```

B.3.7.2. MappingTB_v7_2

```
function [] = MappingTB_v7_2(SigmaOd, SigmaMed) %Sigma(desviación típica) de Odometría y de
    Medida

%Similar a v7_1, pero con otro mapa y diferente base de tiempos.

    LRecinto = 20;                          %lado del recinto cuadrado
    T = 0.5;                                %periodo en segundos
    RadioRobot = 1;                         %Dimensiones del Robot

    %Posición de los Obstaculos
    OBSX = [1  3  9  1  NaN 13 13 14 14 15 15 16 16 13 NaN 16 19 19 16 NaN 10 11 11 12 12 14
        14 10 10];
    OBSY = [12 10 18 12 NaN 19 18 18 17 17 16 16 19 19 NaN 12 14 10 12 NaN 7  7  4  4  7  7  3
          3  7 ];

    %velocidades angulares instantáneas
    Wd = [12*ones(1,5)  11*ones(1,14)   -7*ones(1,5)   12*ones(1,14)   5.9*ones(1,8) 10*ones
        (1,20)];
    Wi = [15*ones(1,5)  11*ones(1,14)    0*ones(1,5)   12*ones(1,14)  -5.9*ones(1,8) 11.5*ones
        (1,20)];

    RecintoX = [0   0          LRecinto    LRecinto    0];
    RecintoY = [0   LRecinto   LRecinto    0           0];

    x = 0;                                      %inicialización de la pose
    y = 0;
    theta = 0;

    r = RadioRobot/6;                       %radio de las ruedas

    %primera iteración en posición inicial y sin error
    [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);
    [MAP,xp,yp,thetap] = Mapping_v6(1,LRecinto,RadioRobot,T,0,0,RAD);

    %estructura de parámetros para mapshow
```

```
        R = maprasterref('XLimWORLD',[0 LRecinto],'YLimWorld',[0 LRecinto],'RasterSize',size(
            MAP));

    mapshow((1-MAP),R);                             %dibuja mapa
    axis equal;
    hold on;
    plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');        %dibuja recinto y obstáculos
    dibujar_v2(RadioRobot,x,y,theta,'g');               %dibuja pose real en verde
    dibujar_v2(RadioRobot,xp,yp,thetap,'k');            %dibuja pose calculada en negro
    hold off;
    pause(0.1);

    Wdp = Wd + SigmaOd*randn(size(Wd));             %incluye error en velocidades angulares
    Wip = Wi + SigmaOd*randn(size(Wi));

    for i = 1:length(Wd)                            %bucle de instantes de tiempo discreto

        Vd = Wd(i)*r;                               %cinemática
        Vi = Wi(i)*r;
        V = (Vd+Vi)/2;
        W = (Vd-Vi)/((10*RadioRobot)/6);

        theta = theta + W*T;
        x = x + V*cos(theta+pi/2)*T;
        y = y + V*sin(theta+pi/2)*T;

        %simulador de telémetro
        [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);

        %incluye error en impactos de telémetro
        RAD = RAD + SigmaMed*randn(size(RAD));

        %mapeo
        [MAP,xp,yp,thetap] = Mapping_v6(0,LRecinto,RadioRobot,T,Wdp(i),Wip(i),RAD);


        mapshow((1-MAP),R);                         %invierte valores y dibuja mapa
        axis equal;
        hold on;
        plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');    %dibuja recinto y obstáculos
        dibujar_v2(RadioRobot,x,y,theta,'g');           %dibuja pose real en verde
        dibujar_v2(RadioRobot,xp,yp,thetap,'k');        %dibuja pose calculada en negro
        hold off;
        pause(0.1);

    end

    DisplayMap(MAP,2);                              %muestra el mapa en figura 2

end
```

B.3.7.3. MappingTB_v7_3

```
function [] = MappingTB_v7_3(SigmaOd, SigmaMed)         %Sigma(desviación típica) de Odometría
     y de Medida

    %Entorno de simulación con un nuevo mapa de obstáculos que además, implementa obstáculos
        moviles con el fin de probar la capacidad de olvido frente a obstáculos no fijos en el
         tiempo.

    LRecinto = 20;                                      %lado del recinto cuadrado
    T = 0.5;                                            %periodo en segundos
    RadioRobot = 1;                                     %Dimensiones del Robot

    %Posición de los Obstaculos fijos
    OBSX = [4   6   6   4   4    NaN    10  14  14  10    10  NaN    9   11  11  9   9
        NaN    2   4   4   2   2   NaN    17  19  19  17  17];
    OBSY = [4   4   2   2   4    NaN    6   6   4   4     6   NaN    13  13  10  10  13
        NaN    16  16  14  14  16  NaN    12  12  8   8   12];
```

```
12
13      %velocidades angulares instantáneas
14      Wd = [12*ones(1,9)  8*ones(1,10)   19.3*ones(1,7)   12*ones(1,9)   -5.2*ones(1,6)  10*ones
            (1,15)   0*ones(1,15)    10*ones(1,18)  ];
15      Wi = [15*ones(1,9)  8*ones(1,10)   12*ones(1,7)    12*ones(1,9)    5.2*ones(1,6)  10*ones
            (1,15)  -6*ones(1,15)    11.2*ones(1,18)];
16
17      RecintoX = [0   0          LRecinto    LRecinto    0];
18      RecintoY = [0   LRecinto   LRecinto    0           0];
19
20      x = 0;                                              %inicialización de la pose
21      y = 0;
22      theta = 0;
23
24      r = RadioRobot/6;                                   %radio de las ruedas
25
26      %estado inicial de los obstáculos móviles
27      OBSXnew = [OBSX    NaN    14  15    15    14    14    NaN    7   8   8   7   7  NaN   9
            10   10   9   9];
28      OBSYnew = [OBSY    NaN    19  19    18    18    19    NaN    12  12  11  11  12 NaN   2
            2    1    1   2];
29
30      %primera iteración en posición inicial y sin error
31      [RAD] = VissionSimulation_v1(OBSXnew,OBSYnew,LRecinto * sqrt(2),x,y,theta);
32      [MAP,xp,yp,thetap] = Mapping_v6(1,LRecinto,RadioRobot,T,0,0,RAD);
33
34      %estructura de parámetros para mapshow
35      R = maprasterref('XLimWORLD',[0 LRecinto],'YLimWorld',[0 LRecinto],'RasterSize',size(MAP))
            ;
36
37      mapshow((1-MAP),R);                                     %dibuja mapa
38      axis equal;
39      hold on;
40      plot(RecintoX,RecintoY,'-.k',OBSXnew,OBSYnew,'r');  %dibuja recinto y obstáculos
41      dibujar_v2(RadioRobot,x,y,theta,'g');               %dibuja pose real en verde
42      dibujar_v2(RadioRobot,xp,yp,thetap,'k');            %dibuja pose calculada en negro
43      hold off;
44      pause(0.1);
45
46      Wdp = Wd + SigmaOd*randn(size(Wd));          %incluye error en velocidades angulares
47      Wip = Wi + SigmaOd*randn(size(Wi));
48
49      for i = 1:length(Wd)                         %bucle de instantes de tiempo discreto
50
51          %actualiza la posición de los obstáculos móviles
52          OBSXnew = [OBSX    NaN    14+5*sin(i/10)    15+5*sin(i/10)    15+5*sin(i/10)    14+5*
                sin(i/10)    14+5*sin(i/10)    NaN    7-(i/5)   8-(i/5)   8-(i/5)   7-(i/5)   7-(
                i/5)  NaN   9+sin(i/5)    10+sin(i/5)    10+sin(i/5)    9+sin(i/5)  9+sin(i/5)
                ];
53          OBSYnew = [OBSY    NaN    19                  19                  18
                18                19          NaN    12          12        11        11
                12    NaN     2                 2               1               1           2
                ];
54
55          %elimina el objeto que se mueva mas alla de -1.5. Funciona solo para objetos de 4
                vertices
56          fuera = find(OBSXnew <= -1.5);
57          if not(isempty(fuera))
58              OBSXnew(fuera(1):(fuera(1)+5)) = NaN;
59          end
60
61          Vd = Wd(i)*r;                            %cinemática
62          Vi = Wi(i)*r;
63          V = (Vd+Vi)/2;
64          W = (Vd-Vi)/((10*RadioRobot)/6);
65
66          theta = theta + W*T;
67          x = x + V*cos(theta+pi/2)*T;
68          y = y + V*sin(theta+pi/2)*T;
69
70          %simulador de telémetro
71          [RAD] = VissionSimulation_v1(OBSXnew,OBSYnew,LRecinto * sqrt(2),x,y,theta);
72
```

```
        %incluye error en impactos de telémetro
        RAD = RAD + SigmaMed*randn(size(RAD));

        %mapeo
        [MAP,xp,yp,thetap] = Mapping_v6(0,LRecinto,RadioRobot,T,Wdp(i),Wip(i),RAD);


        mapshow(1-MAP),R);                                          %dibuja mapa
        axis equal;
        hold on;
        plot(RecintoX,RecintoY,'-.k',OBSXnew,OBSYnew,'r'); %dibuja recinto y obstáculos
        dibujar_v2(RadioRobot,x,y,theta,'g');                %dibuja pose real en verde
        dibujar_v2(RadioRobot,xp,yp,thetap,'k');        %dibuja pose calculada en negro
        hold off;
        pause(0.1);

    end

    DisplayMap(MAP,2);                  %muestra el mapa en figura 2

end
```

B.3.8. MappingTB_v8

B.3.8.1. MappingTB_v8_1

```
function [] = MappingTB_v8_1(SigmaOd, SigmaMed) %Sigma(desviación típica) de Odometría y de
    Medida

%Entorno de simulación para mapping_v7. Sin cambios de simulación.
%descomentar lineas comentadas para generar tabla de error de pose y gráfica.

    LRecinto = 20;                      %lado del recinto cuadrado
    T = 0.1;                            %periodo en segundos
    RadioRobot = 1;                     %Dimensiones del Robot

    %Posición de los Obstaculos
    OBSX = [4 4 6 7 8 6 4 NaN 5  5  6  6  8  8  9  9  5  NaN 12 14 14 12 NaN 13 13 16 16 13
        NaN 12 12 16 16 12 NaN 8  8  11 11 8];
    OBSY = [6 7 9 7 6 7 6 NaN 13 15 15 14 14 15 15 13 13 NaN 14 16 14 14 NaN 11 13 12 11 11
        NaN 7  8  8  7  7  NaN 9  12 12 9  9];

    %velocidades angulares instantáneas
    Wd = [60*ones(1,13)  55*ones(1,10)  60*ones(1,10) 0*ones(1,8)  60*ones(1,15)   33*ones
        (1,7) 50*ones(1,20)];
    Wi = [60*ones(1,13)  72*ones(1,10)  60*ones(1,10) 18*ones(1,8)  60*ones(1,15)  -33*ones
        (1,7) 50*ones(1,20)];

     RecintoX = [0   0           LRecinto    LRecinto    0];
     RecintoY = [0   LRecinto    LRecinto    0           0];

    x = 0;                              %inicialización de la pose
    y = 0;
    theta = 0;

    r = RadioRobot/6;                   %radio de las ruedas

    %primera iteración en posición inicial y sin error
    [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);
    [MAP,xp,yp,thetap] = Mapping_v7(1,LRecinto,RadioRobot,T,0,0,RAD,SigmaOd);

    %estructura de parámetros para mapshow
    R = maprasterref('XLimWORLD',[0 LRecinto],'YLimWorld',[0 LRecinto],'RasterSize',size(MAP))
        ;

    mapshow((1-MAP),R);                                     %dibuja mapa
    axis equal;
    hold on;
```

```
37      plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');  %dibuja recinto y obstáculos
38      dibujar_v2(RadioRobot,x,y,theta,'g');         %dibuja pose real en verde
39      dibujar_v2(RadioRobot,xp,yp,thetap,'k');      %dibuja pose calculada en negro
40      hold off;
41      pause(0.1);
42
43      %incluye error en velocidades angulares
44      Wdp = Wd + SigmaOd*randn(size(Wd));
45      Wip = Wi + SigmaOd*randn(size(Wi));
46
47  %     iteracion = linspace(1,length(Wd)+1,length(Wd)+1)';
48  %     poseX = zeros(length(Wd)+1,1);
49  %     poseY = zeros(length(Wd)+1,1);
50  %     poseTHETA = zeros(length(Wd)+1,1);
51  %     error = zeros(length(Wd)+1,1);
52
53  %     poseX(1) = x;
54  %     poseY(1) = y;
55  %     poseTHETA(1) = theta;
56  %     error(1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);
57
58      for i = 1:length(Wd)          %bucle de instantes de tiempo discreto
59
60          Vd = Wd(i)*r;                                   %cinemática
61          Vi = Wi(i)*r;
62          V = (Vd+Vi)/2;
63          W = (Vd-Vi)/((10*RadioRobot)/6);
64
65          theta = theta + W*T;
66          x = x + V*cos(theta+pi/2)*T;
67          y = y + V*sin(theta+pi/2)*T;
68
69           %simulador de telémetro
70          [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);
71
72          %incluye error en impactos de telémetro
73          RAD = RAD + SigmaMed*randn(size(RAD));
74
75          %mapeo
76          [MAP,xp,yp,thetap] = Mapping_v7(0,LRecinto,RadioRobot,T,Wdp(i),Wip(i),RAD,SigmaOd);
77
78
79          mapshow((1-MAP),R);                             %dibuja mapa
80          axis equal;
81          hold on;
82          plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');  %dibuja recinto y obstáculos
83          dibujar_v2(RadioRobot,x,y,theta,'g');            %dibuja pose real en verde
84          dibujar_v2(RadioRobot,xp,yp,thetap,'k');    %dibuja pose calculada en negro
85          hold off;
86          pause(0.1);
87
88  %         poseX(i+1) = x;
89  %         poseY(i+1) = y;
90  %         poseTHETA(i+1) = theta;
91  %         error(i+1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);
92
93      end
94
95  %     print_table([iteracion round(poseX,3) round(poseY,3) round(poseTHETA,3) round(error,4)
        ],{' %.4g'},'printMode','latex','printLatexFull',false)
96  %     errorcuadraticomedio = mean(error)
97  %     figure(3)
98  %     plot(error,'-s')
99  %     xlabel('Iteración')
100 %     ylabel('Error cuadrático')
101 %     grid on
102     DisplayMap(MAP,2);                  %muestra el mapa en figura 2
103
104 end
```

B.3.8.2. MappingTB_v8_2

```
function [] = MappingTB_v8_2(SigmaOd, SigmaMed) %Sigma(desviación típica) de Odometría y de
    Medida

%Similar a v8_1, pero con otro mapa y diferente base de tiempo.
%descomentar lineas comentadas para generar tabla de error de pose y gráfica.

    LRecinto = 20;              %lado del recinto cuadrado
    T = 0.5;                    %periodo en segundos
    RadioRobot = 1;             %Dimensiones del Robot

    %Posición de los Obstaculos
    OBSX = [1  3  9  1  NaN 13 13 14 14 15 15 16 16 13 NaN 16 19 19 16 NaN 10 11 11 12 12 14
        14 10 10];
    OBSY = [12 10 18 12 NaN 19 18 18 17 17 16 16 19 19 NaN 12 14 10 12 NaN 7  7  4  4  7  7  3
          3  7 ];

    %velocidades angulares instantáneas
    Wd = [12*ones(1,5)  11*ones(1,14)   -7*ones(1,5)   12*ones(1,14)   5.9*ones(1,8) 10*ones
        (1,20)];
    Wi = [15*ones(1,5)  11*ones(1,14)    0*ones(1,5)   12*ones(1,14)  -5.9*ones(1,8) 11.5*ones
        (1,20)];

    RecintoX = [0   0           LRecinto    LRecinto    0];
    RecintoY = [0   LRecinto    LRecinto    0           0];

    x = 0;                      %inicialización de la pose
    y = 0;
    theta = 0;

    r = RadioRobot/6;           %radio de las ruedas

    %primera iteración en posición inicial y sin error
    [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);
    [MAP,xp,yp,thetap] = Mapping_v7(1,LRecinto,RadioRobot,T,0,0,RAD,SigmaOd);

    %estructura de parámetros para mapshow
    R = maprasterref('XLimWORLD',[0 LRecinto],'YLimWorld',[0 LRecinto],'RasterSize',size(MAP))
        ;

    mapshow((1-MAP),R);                                     %dibuja mapa
    axis equal;
    hold on;
    plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r'); %dibuja recinto y obstáculos
    dibujar_v2(RadioRobot,x,y,theta,'g');        %dibuja pose real en verde
    dibujar_v2(RadioRobot,xp,yp,thetap,'k');   %dibuja pose calculada en negro
    hold off;
    pause(0.1);

    %incluye error en velocidades angulares
    Wdp = Wd + SigmaOd*randn(size(Wd));
    Wip = Wi + SigmaOd*randn(size(Wi));

%     iteracion = linspace(1,length(Wd)+1,length(Wd)+1)';
%     poseX = zeros(length(Wd)+1,1);
%     poseY = zeros(length(Wd)+1,1);
%     poseTHETA = zeros(length(Wd)+1,1);
%     error = zeros(length(Wd)+1,1);
%
%     poseX(1) = x;
%     poseY(1) = y;
%     poseTHETA(1) = theta;
%     error(1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);

    for i = 1:length(Wd)                %bucle de instantes de tiempo discreto

        Vd = Wd(i)*r;                   %cinemática
        Vi = Wi(i)*r;
        V = (Vd+Vi)/2;
        W = (Vd-Vi)/((10*RadioRobot)/6);

```

```
        theta = theta + W*T;
        x = x + V*cos(theta+pi/2)*T;
        y = y + V*sin(theta+pi/2)*T;

        %simulador de telémetro
        [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);

        %incluye error en impactos de telémetro
        RAD = RAD + SigmaMed*randn(size(RAD));

        %mapeo
        [MAP,xp,yp,thetap] = Mapping_v7(0,LRecinto,RadioRobot,T,Wdp(i),Wip(i),RAD,SigmaOd);


        mapshow((1-MAP),R);                    %dibuja mapa
        axis equal;
        hold on;
        plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r'); %dibuja recinto y obstáculos
        dibujar_v2(RadioRobot,x,y,theta,'g');        %dibuja pose real en verde
        dibujar_v2(RadioRobot,xp,yp,thetap,'k');   %dibuja pose calculada en negro
        hold off;
        pause(0.1);

%         poseX(i+1) = x;
%         poseY(i+1) = y;
%         poseTHETA(i+1) = theta;
%         error(i+1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);

    end

%     print_table([iteracion round(poseX,3) round(poseY,3) round(poseTHETA,3) round(error,4)
    ],{' %.4g'},'printMode','latex','printLatexFull',false)
%     errorcuadraticomedio = mean(error)
%     figure(3)
%     plot(error,'-s')
%     xlabel('Iteración')
%     ylabel('Error cuadrático')
%     grid on
    DisplayMap(MAP,2);                         %muestra el mapa en figura 2

end
```

B.3.9. MappingTB_v9

B.3.9.1. MappingTB_v9_1

```
function [] = MappingTB_v9_1(SigmaOd, SigmaMed) %Sigma(desviación típica) de Odometría y de
    Medida

%Entorno de simulación para mapping_v8. Sin cambios de simulación.
%descomentar lineas comentadas para generar tabla de error de pose

    LRecinto = 20;              %lado del recinto cuadrado
    T = 0.1;                    %periodo en segundos
    RadioRobot = 1;             %Dimensiones del Robot

    %Posición de los Obstaculos
    OBSX = [4 4 6 7 8 6 4 NaN 5  5  6  6  8  8  9  9  5  NaN 12 14 14 12 NaN 13 13 16 16 13
        NaN 12 12 16 16 12 NaN 8  8  11 11 8];
    OBSY = [6 7 9 7 6 7 6 NaN 13 15 15 14 14 15 15 13 13 NaN 14 16 14 14 NaN 11 13 12 11 11
        NaN 7  8  8  7  7  NaN 9  12 12 9  9];

    %velocidades angulares instantáneas
    Wd = [60*ones(1,13)  55*ones(1,10)  60*ones(1,10) 0*ones(1,8)  60*ones(1,15)   33*ones
        (1,7) 50*ones(1,20)];
    Wi = [60*ones(1,13)  72*ones(1,10)  60*ones(1,10) 18*ones(1,8)  60*ones(1,15)  -33*ones
        (1,7) 50*ones(1,20)];

```

```
18        RecintoX = [0    0           LRecinto    LRecinto    0];
19        RecintoY = [0    LRecinto    LRecinto    0           0];
20
21      x = 0;                              %inicialización de la pose
22      y = 0;
23      theta = 0;
24
25      r = RadioRobot/6;           %radio de las ruedas
26
27      %primera iteración en posición inicial y sin error
28      [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);
29      [MAP,xp,yp,thetap] = Mapping_v8(1,LRecinto,RadioRobot,T,0,0,RAD,SigmaOd);
30
31      %estructura de parámetros para mapshow
32      R = maprasterref('XLimWORLD',[0 LRecinto],'YLimWorld',[0 LRecinto],'RasterSize',size(MAP))
            ;
33
34      mapshow((1-MAP),R);         %dibuja mapa
35      axis equal;
36      hold on;
37      plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');   %dibuja recinto y obstáculos
38      dibujar_v2(RadioRobot,x,y,theta,'g');          %dibuja pose real en verde
39      dibujar_v2(RadioRobot,xp,yp,thetap,'k');    %dibuja pose calculada en negro
40      hold off;
41      pause(0.1);
42
43      %incluye error en velocidades angulares
44      Wdp = Wd + SigmaOd*randn(size(Wd));
45      Wip = Wi + SigmaOd*randn(size(Wi));
46
47  %     iteracion = linspace(1,length(Wd)+1,length(Wd)+1)';
48  %     poseX = zeros(length(Wd)+1,1);
49  %     poseY = zeros(length(Wd)+1,1);
50  %     poseTHETA = zeros(length(Wd)+1,1);
51  %     error = zeros(length(Wd)+1,1);
52  %
53  %     poseX(1) = x;
54  %     poseY(1) = y;
55  %     poseTHETA(1) = theta;
56  %     error(1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);
57
58      for i = 1:length(Wd)        %bucle de instantes de tiempo discreto
59
60          Vd = Wd(i)*r;           %cinemática
61          Vi = Wi(i)*r;
62          V = (Vd+Vi)/2;
63          W = (Vd-Vi)/((10*RadioRobot)/6);
64
65          theta = theta + W*T;
66          x = x + V*cos(theta+pi/2)*T;
67          y = y + V*sin(theta+pi/2)*T;
68
69          %simulador de telémetro
70          [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);
71
72          %incluye error en impactos de telémetro
73          RAD = RAD + SigmaMed*randn(size(RAD));
74
75          %mapeo
76          [MAP,xp,yp,thetap] = Mapping_v8(0,LRecinto,RadioRobot,T,Wdp(i),Wip(i),RAD,SigmaOd);
77
78
79          mapshow((1-MAP),R);                                    %dibuja mapa
80          axis equal;
81          hold on;
82          plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');  %dibuja recinto y obstáculos
83          dibujar_v2(RadioRobot,x,y,theta,'g');             %dibuja pose real en verde
84          dibujar_v2(RadioRobot,xp,yp,thetap,'k');   %dibuja pose calculada en negro
85          hold off;
86          pause(0.1);
87
88  %         poseX(i+1) = x;
89  %         poseY(i+1) = y;
```

```
%         poseTHETA(i+1) = theta;
%         error(i+1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);

    end

%     print_table([iteracion round(poseX,3) round(poseY,3) round(poseTHETA,3) round(error,4)
    ],{' %.4g'},'printMode','latex','printLatexFull',false)
%     errorcuadraticomedio = mean(error)
%     figure(3)
%     plot(error,'-s')
%     xlabel('Iteración')
%     ylabel('Error cuadrático')
%     grid on
    DisplayMap(MAP,2);                  %muestra el mapa en figura 2

end
```

B.3.9.2. MappingTB_v9_2

```
function [] = MappingTB_v9_2(SigmaOd, SigmaMed) %Sigma(desviación típica) de Odometría y de
    Medida

%Similar a v9_1, pero con otro mapa y diferente base de tiempo.
%descomentar lineas comentadas para generar tabla de error de pose y gráfica.

    LRecinto = 20;                      %lado del recinto cuadrado
    T = 0.5;                    %periodo en segundos
    RadioRobot = 1;             %Dimensiones del Robot

    %Posición de los Obstaculos
    OBSX = [1  3  9  1  NaN 13 13 14 14 15 15 16 16 13 NaN 16 19 19 16 NaN 10 11 11 12 12 14
        14 10 10];
    OBSY = [12 10 18 12 NaN 19 18 18 17 17 16 16 19 19 NaN 12 14 10 12 NaN 7  7  4  4  7  7  3
          3  7 ];

        %velocidades angulares instantáneas
        Wd = [12*ones(1,5)  11*ones(1,14)   -7*ones(1,5)   12*ones(1,14)   5.9*ones(1,8) 10*
            ones(1,20)];
    Wi = [15*ones(1,5)  11*ones(1,14)    0*ones(1,5)   12*ones(1,14)  -5.9*ones(1,8) 11.5*ones
        (1,20)];

    RecintoX = [0   0           LRecinto    LRecinto    0];
    RecintoY = [0   LRecinto    LRecinto    0           0];

    x = 0;                      %inicialización de la pose
    y = 0;
    theta = 0;

    r = RadioRobot/6;           %radio de las ruedas

    %primera iteración en posición inicial y sin error
    [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);
    [MAP,xp,yp,thetap] = Mapping_v8(1,LRecinto,RadioRobot,T,0,0,RAD,SigmaOd);

    %estructura de parámetros para mapshow
    R = maprasterref('XLimWORLD',[0 LRecinto],'YLimWorld',[0 LRecinto],'RasterSize',size(MAP))
        ;

    mapshow((1-MAP),R);                                 %dibuja mapa
    axis equal;
    hold on;
    plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r'); %dibuja recinto y obstáculos
    dibujar_v2(RadioRobot,x,y,theta,'g');        %dibuja pose real en verde
    dibujar_v2(RadioRobot,xp,yp,thetap,'k');  %dibuja pose calculada en negro
    hold off;
    pause(0.1);

    %incluye error en velocidades angulares
    Wdp = Wd + SigmaOd*randn(size(Wd));
```

```
45      Wip = Wi + SigmaOd*randn(size(Wi));
46
47  %     iteracion = linspace(1,length(Wd)+1,length(Wd)+1)';
48  %     poseX = zeros(length(Wd)+1,1);
49  %     poseY = zeros(length(Wd)+1,1);
50  %     poseTHETA = zeros(length(Wd)+1,1);
51  %     error = zeros(length(Wd)+1,1);
52  %
53  %     poseX(1) = x;
54  %     poseY(1) = y;
55  %     poseTHETA(1) = theta;
56  %     error(1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);
57
58      for i = 1:length(Wd)                %bucle de instantes de tiempo discreto
59
60          Vd = Wd(i)*r;                   %cinemática
61          Vi = Wi(i)*r;
62          V = (Vd+Vi)/2;
63          W = (Vd-Vi)/((10*RadioRobot)/6);
64
65          theta = theta + W*T;
66          x = x + V*cos(theta+pi/2)*T;
67          y = y + V*sin(theta+pi/2)*T;
68
69          %simulador de telémetro
70          [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);
71
72          %incluye error en impactos de telémetro
73          RAD = RAD + SigmaMed*randn(size(RAD));
74
75          %mapeo
76          [MAP,xp,yp,thetap] = Mapping_v8(0,LRecinto,RadioRobot,T,Wdp(i),Wip(i),RAD,SigmaOd);
77
78
79          mapshow((1-MAP),R);                                          %dibuja mapa
80          axis equal;
81          hold on;
82          plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r'); %dibuja recinto y obstáculos
83          dibujar_v2(RadioRobot,x,y,theta,'g');        %dibuja pose real en verde
84          dibujar_v2(RadioRobot,xp,yp,thetap,'k');  %dibuja pose calculada en negro
85          hold off;
86          pause(0.1);
87
88  %         poseX(i+1) = x;
89  %         poseY(i+1) = y;
90  %         poseTHETA(i+1) = theta;
91  %         error(i+1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);
92
93      end
94
95  %     print_table([iteracion round(poseX,3) round(poseY,3) round(poseTHETA,3) round(error,4)
    ],{' %.4g'},'printMode','latex','printLatexFull',false)
96  %     errorcuadraticomedio = mean(error)
97  %     figure(3)
98  %     plot(error,'-s')
99  %     xlabel('Iteración')
100 %     ylabel('Error cuadrático')
101 %     grid on
102     DisplayMap(MAP,2);                  %muestra el mapa en figura 2
103
104 end
```

B.3.10. MappingTB_v10

B.3.10.1. MappingTB_v10_1

```
function [] = MappingTB_v10_1(SigmaOd, SigmaMed)        %Sigma(desviación típica) de Odometría
    y de Medida

%Entorno de simulación para mapping_v9. Sin cambios de simulación.
%descomentar lineas comentadas para generar tabla de error de pose

    LRecinto = 20;              %lado del recinto cuadrado
    T = 0.1;                    %periodo en segundos
    RadioRobot = 1;             %Dimensiones del Robot

    %Posición de los Obstaculos
    OBSX = [4 4 6 7 8 6 4 NaN 5  5  6  6  8  8  9  9  5  NaN 12 14 14 12 NaN 13 13 16 16 13
        NaN 12 12 16 16 12 NaN 8  8  11 11 8];
    OBSY = [6 7 9 7 6 7 6 NaN 13 15 15 14 14 15 15 13 13 NaN 14 16 14 14 NaN 11 13 12 11 11
        NaN 7  8  8  7  7  NaN 9  12 12 9  9];

    %velocidades angulares instantáneas
    Wd = [60*ones(1,13)  55*ones(1,10)  60*ones(1,10) 0*ones(1,8)  60*ones(1,15)   33*ones
        (1,7) 50*ones(1,20)];
    Wi = [60*ones(1,13)  72*ones(1,10)  60*ones(1,10) 18*ones(1,8)  60*ones(1,15)  -33*ones
        (1,7) 50*ones(1,20)];

     RecintoX = [0   0           LRecinto    LRecinto    0];
     RecintoY = [0   LRecinto    LRecinto    0           0];

    x = 0;                              %inicialización de la pose
    y = 0;
    theta = 0;

    r = RadioRobot/6;           %radio de las ruedas

    %primera iteración en posición inicial y sin error
    [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);
    [MAP,xp,yp,thetap] = Mapping_v9(1,LRecinto,RadioRobot,T,0,0,RAD,SigmaOd);

    %estructura de parámetros para mapshow
    R = maprasterref('XLimWORLD',[0 LRecinto],'YLimWorld',[0 LRecinto],'RasterSize',size(MAP))
        ;

    mapshow((1-MAP),R);         %dibuja mapa
    axis equal;
    hold on;
    plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');   %dibuja recinto y obstáculos
    dibujar_v2(RadioRobot,x,y,theta,'g');          %dibuja pose real en verde
    dibujar_v2(RadioRobot,xp,yp,thetap,'k');    %dibuja pose calculada en negro
    hold off;
    pause(0.1);

    %incluye error en velocidades angulares
    Wdp = Wd + SigmaOd*randn(size(Wd));
    Wip = Wi + SigmaOd*randn(size(Wi));

%     iteracion = linspace(1,length(Wd)+1,length(Wd)+1)';
%     poseX = zeros(length(Wd)+1,1);
%     poseY = zeros(length(Wd)+1,1);
%     poseTHETA = zeros(length(Wd)+1,1);
%     error = zeros(length(Wd)+1,1);
%
%     poseX(1) = x;
%     poseY(1) = y;
%     poseTHETA(1) = theta;
%     error(1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);

    for i = 1:length(Wd)        %bucle de instantes de tiempo discreto

        Vd = Wd(i)*r;           %cinemática
        Vi = Wi(i)*r;
```

```
        V = (Vd+Vi)/2;
        W = (Vd-Vi)/((10*RadioRobot)/6);

        theta = theta + W*T;
        x = x + V*cos(theta+pi/2)*T;
        y = y + V*sin(theta+pi/2)*T;

        %simulador de telémetro
        [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);

        %incluye error en impactos de telémetro
        RAD = RAD + SigmaMed*randn(size(RAD));

        %mapeo
        [MAP,xp,yp,thetap] = Mapping_v9(0,LRecinto,RadioRobot,T,Wdp(i),Wip(i),RAD,SigmaOd);


        mapshow((1-MAP),R);                                          %dibuja mapa
        axis equal;
        hold on;
        plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r'); %dibuja recinto y obstáculos
        dibujar_v2(RadioRobot,x,y,theta,'g');            %dibuja pose real en verde
        dibujar_v2(RadioRobot,xp,yp,thetap,'k');  %dibuja pose calculada en negro
        hold off;
        pause(0.1);

%         poseX(i+1) = x;
%         poseY(i+1) = y;
%         poseTHETA(i+1) = theta;
%         error(i+1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);

    end

%     print_table([iteracion round(poseX,3) round(poseY,3) round(poseTHETA,3) round(error,4)
    ],{' %.4g'},'printMode','latex','printLatexFull',false)
%     errorcuadraticomedio = mean(error)
%     figure(3)
%     plot(error,'-s')
%     xlabel('Iteración')
%     ylabel('Error cuadrático')
%     grid on
    DisplayMap(MAP,2);                  %muestra el mapa en figura 2

end
```

B.3.10.2. MappingTB_v10_2

```
function [] = MappingTB_v10_2(SigmaOd, SigmaMed)        %Sigma(desviación típica) de Odometría
     y de Medida

%Similar a v10_1, pero con otro mapa y diferente base de tiempo.
%descomentar lineas comentadas para generar tabla de error de pose y gráfica.

    LRecinto = 20;                      %lado del recinto cuadrado
    T = 0.5;                    %periodo en segundos
    RadioRobot = 1;             %Dimensiones del Robot

    %Posición de los Obstaculos
    OBSX = [1  3  9  1  NaN 13 13 14 14 15 15 16 16 13 NaN 16 19 19 16 NaN 10 11 11 12 12 14
        14 10 10];
    OBSY = [12 10 18 12 NaN 19 18 18 17 17 16 16 19 19 NaN 12 14 10 12 NaN 7  7  4  4  7  7  3
          3  7 ];

        %velocidades angulares instantáneas
        Wd = [12*ones(1,5)  11*ones(1,14)   -7*ones(1,5)   12*ones(1,14)   5.9*ones(1,8) 10*
            ones(1,20)];
    Wi = [15*ones(1,5)  11*ones(1,14)    0*ones(1,5)   12*ones(1,14)  -5.9*ones(1,8) 11.5*ones
        (1,20)];

```

```
18      RecintoX = [0   0           LRecinto    LRecinto    0];
19      RecintoY = [0   LRecinto    LRecinto    0           0];
20
21      x = 0;                      %inicialización de la pose
22      y = 0;
23      theta = 0;
24
25      r = RadioRobot/6;           %radio de las ruedas
26
27      %primera iteración en posición inicial y sin error
28      [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);
29      [MAP,xp,yp,thetap] = Mapping_v9(1,LRecinto,RadioRobot,T,0,0,RAD,SigmaOd);
30
31      %estructura de parámetros para mapshow
32      R = maprasterref('XLimWORLD',[0 LRecinto],'YLimWorld',[0 LRecinto],'RasterSize',size(MAP))
            ;
33
34      mapshow((1-MAP),R);                                 %dibuja mapa
35      axis equal;
36      hold on;
37      plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');  %dibuja recinto y obstáculos
38      dibujar_v2(RadioRobot,x,y,theta,'g');         %dibuja pose real en verde
39      dibujar_v2(RadioRobot,xp,yp,thetap,'k');   %dibuja pose calculada en negro
40      hold off;
41      pause(0.1);
42
43      %incluye error en velocidades angulares
44      Wdp = Wd + SigmaOd*randn(size(Wd));
45      Wip = Wi + SigmaOd*randn(size(Wi));
46
47  %     iteracion = linspace(1,length(Wd)+1,length(Wd)+1)';
48  %     poseX = zeros(length(Wd)+1,1);
49  %     poseY = zeros(length(Wd)+1,1);
50  %     poseTHETA = zeros(length(Wd)+1,1);
51  %     error = zeros(length(Wd)+1,1);
52  %
53  %     poseX(1) = x;
54  %     poseY(1) = y;
55  %     poseTHETA(1) = theta;
56  %     error(1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);
57
58      for i = 1:length(Wd)                %bucle de instantes de tiempo discreto
59
60          Vd = Wd(i)*r;                   %cinemática
61          Vi = Wi(i)*r;
62          V = (Vd+Vi)/2;
63          W = (Vd-Vi)/((10*RadioRobot)/6);
64
65          theta = theta + W*T;
66          x = x + V*cos(theta+pi/2)*T;
67          y = y + V*sin(theta+pi/2)*T;
68
69          %simulador de telémetro
70          [RAD] = VissionSimulation_v1(OBSX,OBSY,LRecinto * sqrt(2),x,y,theta);
71
72          %incluye error en impactos de telémetro
73          RAD = RAD + SigmaMed*randn(size(RAD));
74
75          %mapeo
76          [MAP,xp,yp,thetap] = Mapping_v9(0,LRecinto,RadioRobot,T,Wdp(i),Wip(i),RAD,SigmaOd);
77
78
79          mapshow((1-MAP),R);                                 %dibuja mapa
80          axis equal;
81          hold on;
82          plot(RecintoX,RecintoY,'-.k',OBSX,OBSY,'r');  %dibuja recinto y obstáculos
83          dibujar_v2(RadioRobot,x,y,theta,'g');         %dibuja pose real en verde
84          dibujar_v2(RadioRobot,xp,yp,thetap,'k');   %dibuja pose calculada en negro
85          hold off;
86          pause(0.1);
87
88  %         poseX(i+1) = x;
89  %         poseY(i+1) = y;
```

```
90  %         poseTHETA(i+1) = theta;
91  %         error(i+1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);
92
93      end
94
95  %     print_table([iteracion round(poseX,3) round(poseY,3) round(poseTHETA,3) round(error,4)
        ],{' %.4g'},'printMode','latex','printLatexFull',false)
96  %     errorcuadraticomedio = mean(error)
97  %     figure(3)
98  %     plot(error,'-s')
99  %     xlabel('Iteración')
100 %     ylabel('Error cuadrático')
101 %     grid on
102     DisplayMap(MAP,2);                  %muestra el mapa en figura 2
103
104 end
```

B.3.10.3. MappingTB_v10_3

```
1   function [] = MappingTB_v10_3(SigmaOd, SigmaMed)        %Sigma(desviación típica) de Odometría
        y de Medida
2
3   %Entorno de simulación con un nuevo mapa de obstáculos que además, implementa obstáculos
        moviles con el fin de probar la capacidad de olvido frente a obstáculos no fijos en el
        tiempo.
4   %descomentar lineas comentadas para generar tabla de error de pose y gráfica.
5
6       LRecinto = 20;              %lado del recinto cuadrado
7       T = 0.5;                    %periodo en segundos
8       RadioRobot = 1;             %Dimensiones del Robot
9
10      %Posición de los Obstaculos fijos
11      OBSX = [4   6   6   4   4    NaN     10  14  14  10     10  NaN     9   11  11  9   9
            NaN     2   4   4   2   2   NaN     17  19  19  17  17];
12      OBSY = [4   4   2   2   4    NaN     6   6   4   4      6   NaN     13  13  10  10  13
            NaN     16  16  14  14  16  NaN     12  12  8   8   12];
13
14      %velocidades angulares instantáneas
15      Wd = [12*ones(1,9)  8*ones(1,10)   19.3*ones(1,7)   12*ones(1,9)   -5.2*ones(1,6)  10*ones
            (1,15)   0*ones(1,15)    10*ones(1,18)  ];
16      Wi = [15*ones(1,9)  8*ones(1,10)   12*ones(1,7)    12*ones(1,9)    5.2*ones(1,6)  10*ones
            (1,15)  -6*ones(1,15)    11.2*ones(1,18)];
17
18      RecintoX = [0   0           LRecinto    LRecinto    0];
19      RecintoY = [0   LRecinto    LRecinto    0           0];
20
21      x = 0;                      %inicialización de la pose
22      y = 0;
23      theta = 0;
24
25      r = RadioRobot/6;           %radio de las ruedas
26
27       %estado inicial de los obstáculos móviles
28      OBSXnew = [OBSX    NaN    14  15    15    14    14    NaN     7   8   8   7   7  NaN   9
            10   10   9   9];
29      OBSYnew = [OBSY    NaN    19  19    18    18    19    NaN     12  12  11  11  12 NaN   2
            2    1    1   2];
30
31      %primera iteración en posición inicial y sin error
32      [RAD] = VissionSimulation_v1(OBSXnew,OBSYnew,LRecinto * sqrt(2),x,y,theta);
33      [MAP,xp,yp,thetap] = Mapping_v9(1,LRecinto,RadioRobot,T,0,0,RAD,SigmaOd);
34
35
36      %estructura de parámetros para mapshow
37      R = maprasterref('XLimWORLD',[0 LRecinto],'YLimWorld',[0 LRecinto],'RasterSize',size(MAP))
            ;
38
39      mapshow((1-MAP),R);                                                  %dibuja mapa
40      axis equal;
```

```
41      hold on;
42      plot(RecintoX,RecintoY,'-.k',OBSXnew,OBSYnew,'r'); %dibuja recinto y obstáculos
43      dibujar_v2(RadioRobot,x,y,theta,'g');              %dibuja pose real en verde
44      dibujar_v2(RadioRobot,xp,yp,thetap,'k');       %dibuja pose calculada en negro
45      hold off;
46      pause(0.1);
47
48      %incluye error en velocidades angulares
49      Wdp = Wd + SigmaOd*randn(size(Wd));
50      Wip = Wi + SigmaOd*randn(size(Wi));
51
52  %     iteracion = linspace(1,length(Wd)+1,length(Wd)+1)';
53  %     poseX = zeros(length(Wd)+1,1);
54  %     poseY = zeros(length(Wd)+1,1);
55  %     poseTHETA = zeros(length(Wd)+1,1);
56  %     error = zeros(length(Wd)+1,1);
57  %
58  %     poseX(1) = x;
59  %     poseY(1) = y;
60  %     poseTHETA(1) = theta;
61  %     error(1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);
62
63      for i = 1:length(Wd)                %bucle de instantes de tiempo discreto
64
65          %actualiza la posición de los obstáculos móviles
66          OBSXnew = [OBSX    NaN    14+5*sin(i/10)    15+5*sin(i/10)    15+5*sin(i/10)    14+5*
                sin(i/10)    14+5*sin(i/10)    NaN     7-(i/5)   8-(i/5)   8-(i/5)   7-(i/5)   7-(
                i/5)  NaN   9+sin(i/5)     10+sin(i/5)     10+sin(i/5)     9+sin(i/5)  9+sin(i/5)
                ];
67          OBSYnew = [OBSY    NaN     19                    19                   18
                18                 19           NaN     12          12        11        11
                12    NaN      2                 2               1               1           2
                ];
68
69          %elimina el objeto que se mueva mas alla de -1.5. Funciona solo para objetos de 4
                vertices
70          fuera = find(OBSXnew <= -1.5);
71          if not(isempty(fuera))
72              OBSXnew(fuera(1):(fuera(1)+5)) = NaN;
73          end
74
75          Vd = Wd(i)*r;                           %cinemática
76          Vi = Wi(i)*r;
77          V = (Vd+Vi)/2;
78          W = (Vd-Vi)/((10*RadioRobot)/6);
79
80          theta = theta + W*T;
81          x = x + V*cos(theta+pi/2)*T;
82          y = y + V*sin(theta+pi/2)*T;
83
84          %simulador de telémetro
85          [RAD] = VissionSimulation_v1(OBSXnew,OBSYnew,LRecinto * sqrt(2),x,y,theta);
86
87          %incluye error en impactos de telémetro
88          RAD = RAD + SigmaMed*randn(size(RAD));
89
90          %mapeo
91          [MAP,xp,yp,thetap] = Mapping_v9(0,LRecinto,RadioRobot,T,Wdp(i),Wip(i),RAD,SigmaOd);
92
93
94          mapshow((1-MAP),R);                                                 %dibuja mapa
95          axis equal;
96          hold on;
97          plot(RecintoX,RecintoY,'-.k',OBSXnew,OBSYnew,'r'); %dibuja recinto y obstáculos
98          dibujar_v2(RadioRobot,x,y,theta,'g');              %dibuja pose real en verde
99          dibujar_v2(RadioRobot,xp,yp,thetap,'k');       %dibuja pose calculada en negro
100         hold off;
101         pause(0.1);
102
103 %         poseX(i+1) = x;
104 %         poseY(i+1) = y;
105 %         poseTHETA(i+1) = theta;
106 %         error(i+1) = sqrt((x-xp)^2+(y-yp)^2+(theta-thetap)^2);
```

```
107
108         end
109
110     %     print_table([iteracion round(poseX,3) round(poseY,3) round(poseTHETA,3) round(error,4)
        ],{' %.4g'},'printMode','latex','printLatexFull',false)
111     %     errorcuadraticomedio = mean(error)
112     %     figure(3)
113     %     plot(error,'-s')
114     %     xlabel('Iteración')
115     %     ylabel('Error cuadrático')
116     %     grid on
117         DisplayMap(MAP,2);                  %muestra el mapa en figura 2
118
119     end
```

Apéndice C

Funciones Complementarias

C.1. dibujar_v1

La función *dibujar_v1* dibuja un robot azul en la figura actual. Este cuenta, además de la figura del robot, un círculo rojo que indica la cabecera y dos líneas discontinuas que indican el rango de visión del telémetro. La función recibe como argumentos de entrada el radio del robot (R) y las tres variables referentes a la pose (x,y,theta). Para poder dibujar el robot sin borrar lo que haya en la figura, debe haber un *hold on* previo a su llamada.

El algoritmo define una serie de puntos para cada elemento del dibujo, el número de puntos por elemento se puede modificar para obtener mayor o menor resolución en los círculos. Posteriormente, se hace una transformación al *sistema de referencia global* haciendo uso de la pose y finalmente, se dibujan todos los elementos.

A continuación un ejemplo de su funcionamiento y el código correspondiente.

```
1 dibujar_v1(5,10,10,-pi/4)
```

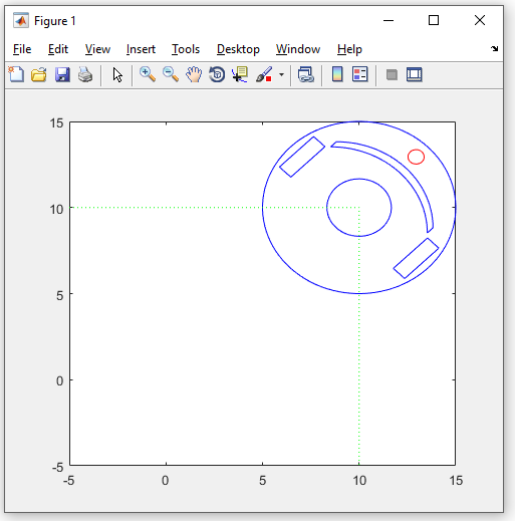


Figura C.1: Ejemplo de funcionamiento de *dibujar_v1*

```
function [] = dibujar_v1(R,x,y,alpha)

alpha = alpha + (pi/2);             %corrección de orientación

AngInicial = 225;
AngFinal   = -45;
puntos = 100;
j = 1;
for i = 0:(2*pi/puntos):2*pi        %puntos de los circulos
    ceX(j)= R*cos(i);               %circulo externo
    ceY(j)= R*sin(i);
    ciX(j)= R/3*cos(i);             %circulo interno
    ciY(j)= R/3*sin(i);
    c1X(j)= 5*R/6 + R/12*cos(i);    %circulo de cabecera
    c1Y(j)= R/12*sin(i);
    j = j+1;
end

j=1;
k = puntos+1;
for i= (-pi/4):(pi/(2*puntos)):(pi/4)         %puntos de los SemiCirculos
    sceX(j) = -4*R/12 + R*cos(i);
    sceY(j) = R*sin(i);
    sciX(k) = -5*R/12 + R*cos(i);
    sciY(k) = R*sin(i);
    j = j+1;
    k = k-1;
end

scX = [sceX sciX sceX(1)];          %vectores para dibujar
scY = [sceY sciY sceY(1)];
r1X=[-R/4 +R/4 +R/4 -R/4 -R/4];     %ruedas
r1Y=[5*R/6+R/12 5*R/6+R/12 5*R/6-R/12 5*R/6-R/12 5*R/6+R/12];
r2X=[-R/4 +R/4 +R/4 -R/4 -R/4];
r2Y=[-5*R/6+R/12 -5*R/6+R/12 -5*R/6-R/12 -5*R/6-R/12 -5*R/6+R/12];

%líneas del campo de visión
vissionX = [3*R*cos((AngInicial-90)*pi/180) 0 3*R*cos((AngFinal-90)*pi/180)];
vissionY = [3*R*sin((AngInicial-90)*pi/180) 0 3*R*sin((AngFinal-90)*pi/180)];

T = [cos(alpha)     -sin(alpha)     x;          %matriz de transformación
     sin(alpha)     cos(alpha)      y;
     0              0               1 ];

ce = [ceX;                                      %formato y transformación de S.R.
       ceY;
       ones(size(ceX))];
ci = [ciX;
       ciY;
       ones(size(ciX))];
c1 = [c1X;
       c1Y;
       ones(size(c1X))];
sc = [scX;
       scY;
       ones(size(scX))];
r1 = [r1X;
       r1Y;
       ones(size(r1X))];
r2 = [r2X;
       r2Y;
       ones(size(r2X))];

vission = [vissionX;
           vissionY;
           ones(size(vissionX))];


cep = T * ce;
cip = T * ci;
c1p = T * c1;
scp = T * sc;
```

```
73  r1p = T * r1;
74  r2p = T * r2;
75
76  vissionp = T * vission;
77
78  cepX = cep(1,:);
79  cepY = cep(2,:);
80  cipX = cip(1,:);
81  cipY = cip(2,:);
82  c1pX = c1p(1,:);
83  c1pY = c1p(2,:);
84  scpX = scp(1,:);
85  scpY = scp(2,:);
86  r1pX = r1p(1,:);
87  r1pY = r1p(2,:);
88  r2pX = r2p(1,:);
89  r2pY = r2p(2,:);
90
91  vissionpX = vissionp(1,:);
92  vissionpY = vissionp(2,:);
93
94   %dibujar
95  plot(cepX,cepY,'b',cipX,cipY,'b',c1pX,c1pY,'r',scpX,scpY,'b',r1pX,r1pY,'b',r2pX,r2pY,'b',
        vissionpX,vissionpY,':g');
96
97  end
```

C.2. dibujar_v2

La función *dibujar_v2* es similar a su antecesora, con la pequeña modificación de permitir elegir el color del dibujo mediante una de las variables de entrada, este se debe expresar en el formato de color de Matlab ('b', 'r', 'g', 'y', 'k', etc).

A continuación dos ejemplos de su funcionamiento y el código correspondiente.

```
1 dibujar_v2(5,10,10,-pi/4,'r')
```

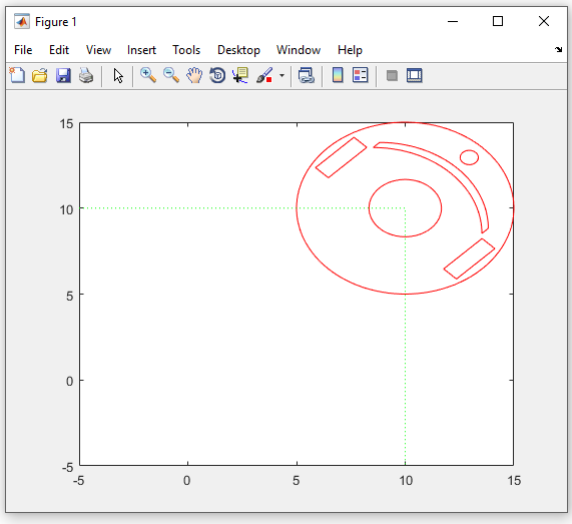


Figura C.2: Ejemplo 1 de funcionamiento de *dibujar_v2*

```
1 dibujar_v2(5,10,10,-pi/4,'b')
```

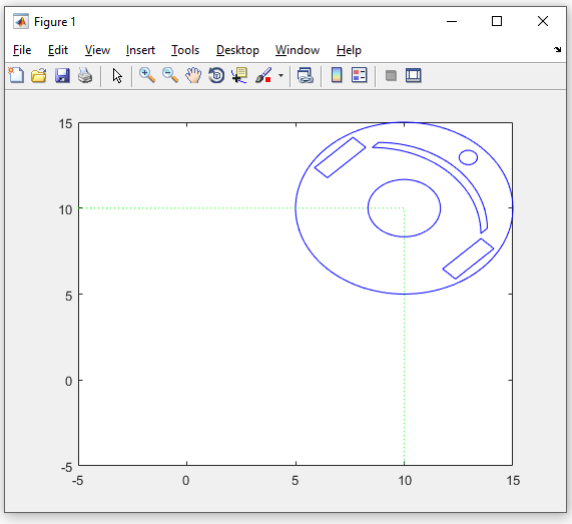


Figura C.3: Ejemplo 2 de funcionamiento de *dibujar_v2*

```
function [] = dibujar_v2(R,x,y,alpha,s)

alpha = alpha + (pi/2);                    %correccion de orientacion

AngInicial = 225;
AngFinal   = -45;
puntos = 100;                              %puntos a dibujar de los circulos
j = 1;
for i = 0:(2*pi/puntos):2*pi
    ceX(j)= R*cos(i);                      %circulo externo
    ceY(j)= R*sin(i);
    ciX(j)= R/3*cos(i);                    %circulo interno
    ciY(j)= R/3*sin(i);
    c1X(j)= 5*R/6 + R/12*cos(i);           %circulo de cabecera
    c1Y(j)= R/12*sin(i);
    j = j+1;
end

j=1;
k = puntos+1;
for i= (-pi/4):(pi/(2*puntos)):(pi/4)      %puntos de los semicirculos
    sceX(j) = -4*R/12 + R*cos(i);
    sceY(j) = R*sin(i);
    sciX(k) = -5*R/12 + R*cos(i);
    sciY(k) = R*sin(i);
    j = j+1;
    k = k-1;
end

scX = [sceX sciX sceX(1)];                 %vectores para dibujar
scY = [sceY sciY sceY(1)];
r1X=[-R/4 +R/4 +R/4 -R/4 -R/4];            %ruedas
r1Y=[5*R/6+R/12 5*R/6+R/12 5*R/6-R/12 5*R/6-R/12 5*R/6+R/12];
r2X=[-R/4 +R/4 +R/4 -R/4 -R/4];
r2Y=[-5*R/6+R/12 -5*R/6+R/12 -5*R/6-R/12 -5*R/6-R/12 -5*R/6+R/12];

 %lineas de campo de vision
vissionX = [3*R*cos((AngInicial-90)*pi/180) 0 3*R*cos((AngFinal-90)*pi/180)];
vissionY = [3*R*sin((AngInicial-90)*pi/180) 0 3*R*sin((AngFinal-90)*pi/180)];

T = [cos(alpha)      -sin(alpha)      x;        %matriz de transformación
     sin(alpha)      cos(alpha)       y;
     0               0                1 ];

ce = [ceX;                                     %formato y transformación de S.R.
       ceY;
       ones(size(ceX))];
ci = [ciX;
       ciY;
       ones(size(ciX))];
c1 = [c1X;
       c1Y;
       ones(size(c1X))];
sc = [scX;
       scY;
       ones(size(scX))];
r1 = [r1X;
       r1Y;
       ones(size(r1X))];
r2 = [r2X;
       r2Y;
       ones(size(r2X))];

vission = [vissionX;
           vissionY;
           ones(size(vissionX))];


cep = T * ce;
cip = T * ci;
c1p = T * c1;
scp = T * sc;
```

```
r1p = T * r1;
r2p = T * r2;

vissionp = T * vission;

cepX = cep(1,:);
cepY = cep(2,:);
cipX = cip(1,:);
cipY = cip(2,:);
c1pX = c1p(1,:);
c1pY = c1p(2,:);
scpX = scp(1,:);
scpY = scp(2,:);
r1pX = r1p(1,:);
r1pY = r1p(2,:);
r2pX = r2p(1,:);
r2pY = r2p(2,:);

vissionpX = vissionp(1,:);
vissionpY = vissionp(2,:);

 %dibujar
plot(cepX,cepY,s,cipX,cipY,s,c1pX,c1pY,s,scpX,scpY,s,r1pX,r1pY,s,r2pX,r2pY,s,vissionpX,
    vissionpY,':g');

end
```

C.3. print_table

Función de la comunidad de Matlab[1] que genera tablas en distintos formatos a partir de vectores o matrices. Para nuestro caso hemos generado las tablas en formato LaTEX para poder incluirlas en este documento.

C.4. CalculateMAP_v1

Esta función se encarga de generar el mapa discreto a partir de la información recibida del telémetro láser. El mapa a calcular será una *Occupancy Grid*, que como ya se explicó en el apartado 3.2.1, trata de una rejilla de posiciones donde cada posición puede tener o no un obstáculo. Así, se necesita definir unos parámetros características de la rejilla correspondientes a sus dimensiones externas (L) y a las dimensiones de sus celdas (Δ).

Para que un punto sea considerado dentro de la celda, debe ser mayor igual que su límite inferior y menor estricto que su límite superior en cada una de sus dimensiones. Esto se hace así por facilitar los cálculos de la discretización, de los que se habla más adelante.

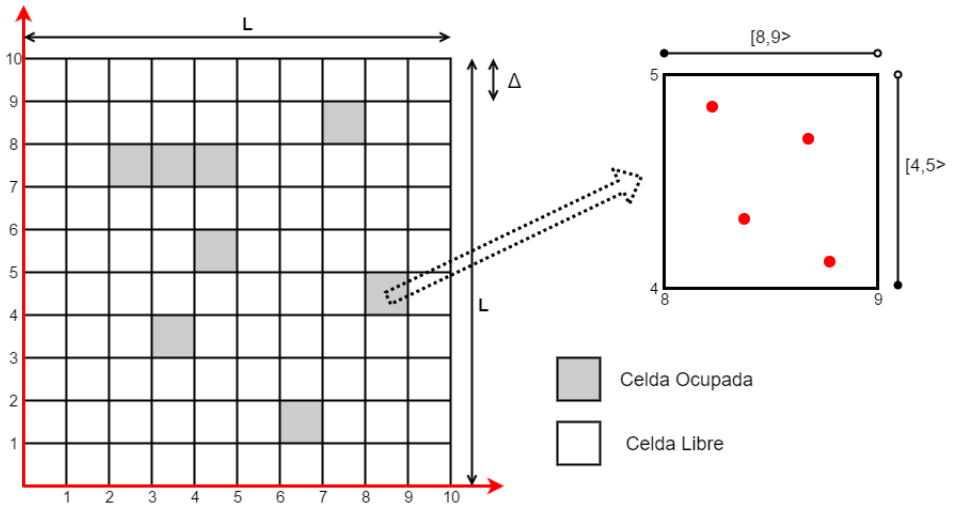
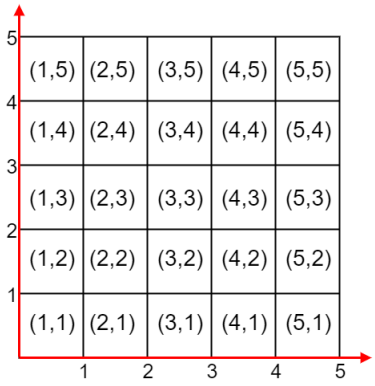


Figura C.4: Occupancy Grid

La definición informática de la *Occupancy Grid* se ha hecho por medio de una matriz bidimensional donde cada elemento se corresponde a una celda, y tendrá valor 1 o 0 en función de haber o no un obstáculo en dicha celda. Respecto al formato de almacenamiento,el número de fila se corresponde a la coordenada Y de la celda y el número de columna a la coordenada X, es decir que visualmente, la matriz es similar a la rejilla pero con el eje Y invertido. Este formato de matriz se ha elegido así para facilitar su representación gráfica haciendo uso de la función *mapshow*.



$$MAP = \begin{bmatrix} (1,1) & (2,1) & (3,1) & (4,1) & (5,1) \\ (1,2) & (2,2) & (3,2) & (4,2) & (5,2) \\ (1,3) & (2,3) & (3,3) & (4,3) & (5,3) \\ (1,4) & (2,4) & (3,4) & (4,4) & (5,4) \\ (1,5) & (2,5) & (3,5) & (4,5) & (5,5) \end{bmatrix}$$

Para hallar los indices de la celda (x', y') a la que pertenece un punto de coordenadas (x, y) aplicamos las siguientes fórmulas:

$$x' = \left\lfloor \frac{x}{p} \right\rfloor + 1 \tag{C.1}$$

$$y' = \left\lfloor \frac{y}{p} \right\rfloor + 1 \tag{C.2}$$

Recordemos que $\lfloor x \rfloor$ representa a la función piso, que redondea a x por defecto, es decir, coge la parte entera de este.
Luego, para dar con el correspondiente elemento de la matriz, solo hace falta poner el valor y' como primer indice de esta y el valor x' como segundo. Por ejemplo, en caso de tratarse de un obstáculo que queremos registrar, el comando sería:

```
1 MAP(y',x') = 1;
```

Bien, una vez definida la lógica y la metodología a seguir, se implementa la función correspondiente. Características principales de la misma:

- Acepta como variables de entrada la dimensión del recinto(*LongRes*), dimensión de las celdas(*Incremento*), pose del robot(*x,y,theta*) y el vector de impactos de telémetro(*RAD*).
- Devuelve como resultado la matriz correspondiente al mapa discreto o *Occupancy Grid*.
- Al igual que su antecesora analógica, tiene definidas como variables internas los parámetros correspondientes al telémetro, pudiendo modificarse según la necesidad.

Asimismo, el procedimiento seguido por la misma es:

1. Al igual que para su antecesor analógico, se calcula el mapa continuo mediante cambio de coordenadas polares a cartesianas y transformación de sistema de referencia, obteniendo un conjunto de pares ordenados correspondientes a los impactos positivos del telémetro.
2. Se define la matriz del mapa discreto con sus respectivas dimensiones y todos sus elementos con valor 0.
3. Una vez obtenidos los pares ordenados del mapa continuo, se realiza un bucle donde para cada par ordenado (siempre y cuando sea un impacto valido), se calcula los índices de la celda discreta en la que se encuentra y a continuación, se fuerza el valor del correspondiente elemento de la matriz a 1.
4. Se devuelve dicha matriz como salida de la función.

El resultado de dicha función se puede consultar en el apartado 3.2.1, en el subapartado *Entorno de simulación*. El código de la función se detalla a continuación. Cabe destacar que se ha implementado también una función que dibuja el mapa a partir de este formato de matriz. Dicha función se ha llamado *DisplayMap* y se puede consultar en este mismo apéndice.

```
function [MAPdisc] = CalculateMAP_v1(LongRes,Incremento,x,y,theta,RAD)


    AngInicial = 225;                       %parámatros del telémetro
    Paso = 0.5;                             %valor positivo
    AngFinal = -45;


    res = fix(LongRes / Incremento);        %número de celdas en cada dimensión de la rejilla

    MAPdisc = zeros(res,res);               %se define el mapa discreto

    NumMedidas = abs((AngInicial - AngFinal)/Paso) + 1;

    if AngInicial > AngFinal                %corrige signo del paso
        Paso = -Paso;
    end

    if length(RAD) ~= NumMedidas
        error('dimensiones incorrectas del vector de medidas');
    end

    ANG = AngInicial : Paso : AngFinal;   %vector de angulos

    MapPartX = zeros(1,NumMedidas);       %inicializamos vectores
    MapPartY = zeros(1,NumMedidas);

    for i = 1 : NumMedidas                %bucle de identificacion de distancias
        MapPartX(i) = RAD(i) * cos(ANG(i)*pi/180);
        MapPartY(i) = RAD(i) * sin(ANG(i)*pi/180);
    end

    MapPart = [ MapPartX;                 %formato y transformación de SR
                MapPartY;
                ones(size(MapPartX))];

    T01 = [ cos(theta)  -sin(theta)       x;
            sin(theta)   cos(theta)       y;
            0            0                1 ];

    MAPcont = T01 * MapPart;

    MAPcontX = MAPcont(1,:);
    MAPcontY = MAPcont(2,:);

    for i = 1 : NumMedidas              %Bucle para cada par ordenado
        if not(isnan(MAPcontX(i)) || isnan(MAPcontY(i)))    %Solo continua si el impacto es no
            nulo
            xp = fix(MAPcontX(i)/Incremento) + 1;           %Se ubica la celda
            yp = fix(MAPcontY(i)/Incremento) + 1;
            MAPdisc(yp,xp) = 1;                             %se fuerza su valor a 1
        end
    end
end
```

C.5. DisplayMap

Esta función tiene como objetivo dibujar los mapas de posiciones discretas usados en este trabajo, ya sean de valores absolutos o probabilísticos siempre y cuando estos se encuentren entre 0 y 1. Recibe como argumentos de entrada el mapa en forma de matriz[1] y como entrada opcional, el número de figura donde se desea mostrar el mapa; si no se pasa dicho argumento, el mapa se muestra en la figura 1.

Su procedimiento lógico es:

1. Inicializa una matriz donde guardará la imagen a mostrar.

2. Traslada los elementos del mapa a la matriz cambiando el orden de las mismas para adaptarlo al formato de imagen. Es decir que invierte el orden de los elementos de cada columna.

3. Corrige los valores de la matriz para que el 0 del mapa (sin obstáculo) sea un píxel blanco (1 en la matriz) y el 1 del mapa (obstáculo seguro) sea un píxel blanco (0 en la matriz). Para ello hace '$1 - valor$'.

4. Dibuja la imagen en la figura seleccionada haciendo uso de la función *imshow*.

A continuación se muestran dos ejemplos del resultado y el código correspondiente.

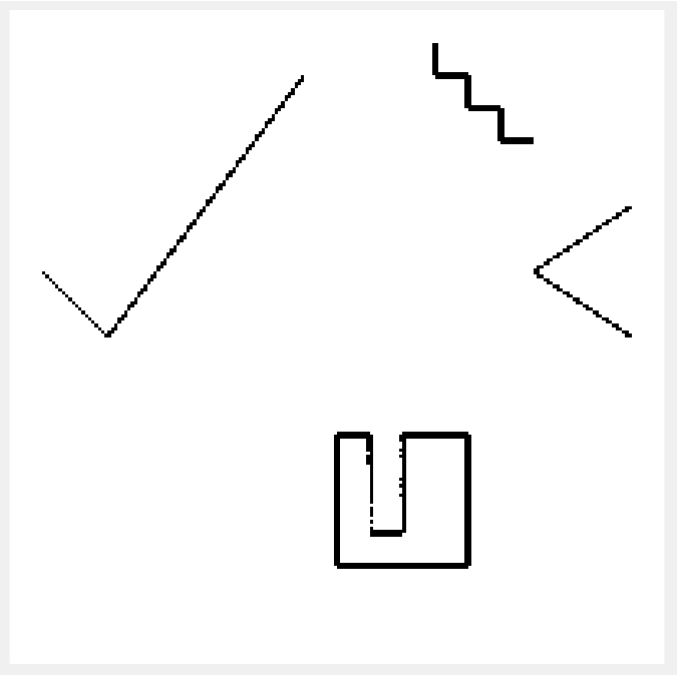


Figura C.5: Resultado de *DisplayMap* con mapa booleano

[1] Las dimensiones de la matriz no están definidas, es decir que la función se adapta a estas.

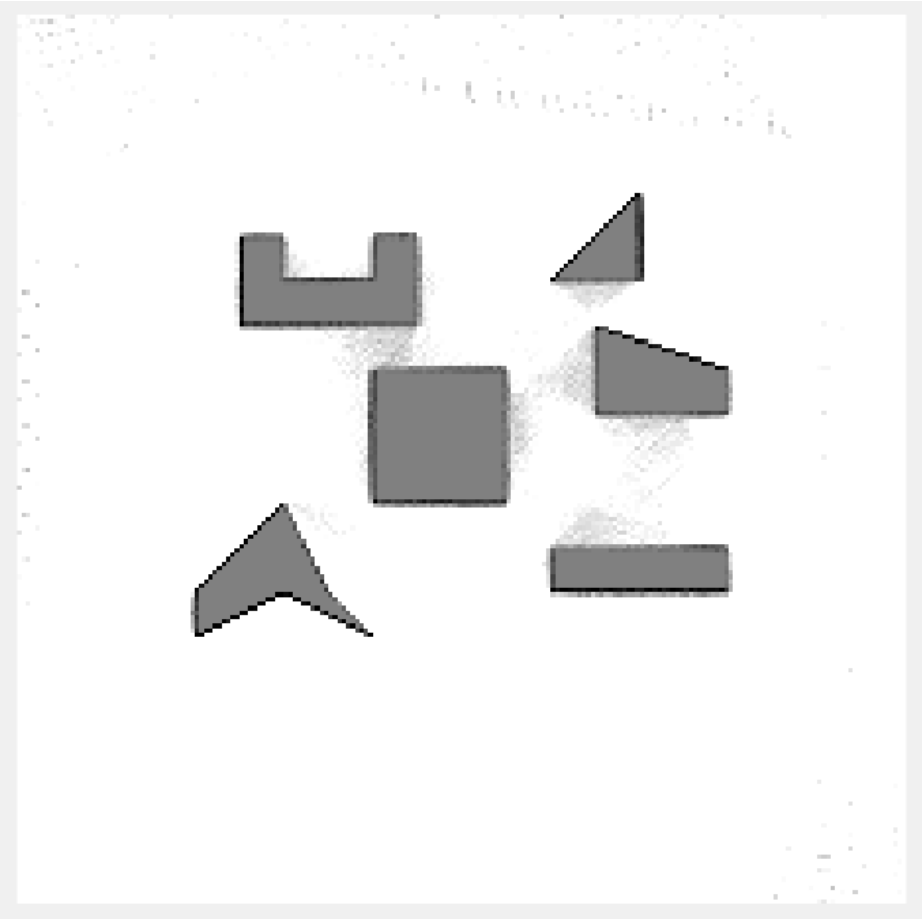


Figura C.6: Resultado de *DisplayMap* con mapa probabilístico

```
function [] = DisplayMap(MAP, fig)
%Prepara el mapa para ser mostrado como imagen y lo muestra en la figura
%elegida, si no recibe figura, usa la 1 por defecto

%valor por defecto de fig
    if nargin == 1
        fig = 1;
    end

    SHOW = zeros(size(MAP));

%corrige posiciones

    for i = 1:size(MAP,1)
        for j = 1:size(MAP,2)
            SHOW(i,j) = MAP(size(MAP,1)-i+1,j);
        end
    end

%corrige valores

    SHOW = ones(size(SHOW)) - SHOW;

%dibuja

    figure(fig);
    imshow(SHOW);

end
```

C.6. CalculateMAP_v2

Esta función es muy similar a su predecesora salvo por ligeras modificaciones que implementa con el objetivo de mejorar el rendimiento. La primera mejora cambia el método de hallar las coordenadas de un impacto a partir de las coordenadas polares recibidas del telémetro. El método que se usaba hasta ahora fue introducido en *Mapping_v1* y su funcionamiento puede consultarse en el apartado 2.1. El nuevo método propone resolver este problema de forma geométrica:

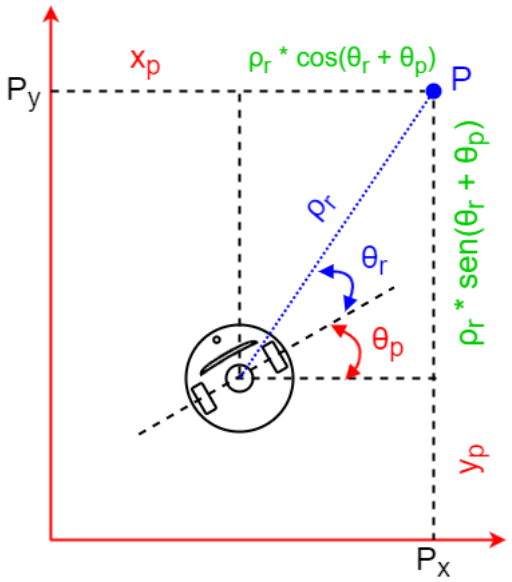


Figura C.7: Metodo geométrico de *CalculateMAP_v2*

$$P_x = x_p + \rho_r \cdot cos(\theta_p + \theta_r) \tag{C.3}$$

$$P_y = y_p + \rho_r \cdot sen(\theta_p + \theta_r) \tag{C.4}$$

Donde:

P: Punto del impacto que se quiere calcular
ρ_r, θ_r: Coordenadas polares del punto P
x_p, y_p, θ_p: Pose del robot
P_x, P_y: Coordenadas cartesianas del punto P

La segunda mejora propuesta es una comprobación adicional previa a discretizar el punto, se comprueba que las coordenadas estén dentro de los límites del recinto. Esto se hace ya que la versión anterior daba errores cuando por alguna circunstancia las coordenadas de un impacto estaban fuera del recinto, debido a que el indice que buscaba en la *Occupancy Grid* no existía realmente al estar esta acotada al recinto.

A continuación se observa los resultados temporales de esta nueva versión y de su predecesora, demostrando la mejora de rendimiento debido al menor coste computacional.

Profile Summary

Generated 27-Aug-2016 18:13:53 using performance time.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
MappingTB_v5_1	1	44.377 s	8.500 s	
VissionSimulation_v1	84	33.491 s	0.679 s	
Mapping_v4	84	0.049 s	0.017 s	
dibujar_v2	168	0.524 s	0.445 s	
CalculateMAP_v1	84	0.032 s	0.032 s	

Figura C.8: Desempeño temporal de *CalculateMAP_v1*

Profile Summary

Generated 27-Aug-2016 18:10:41 using performance time.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
MappingTB_v6_1	1	44.935 s	8.500 s	
VissionSimulation_v1	84	33.482 s	0.686 s	
Mapping_v5	84	0.601 s	0.220 s	
CalculateMAP_v2	2325	0.270 s	0.270 s	
dibujar_v2	168	0.522 s	0.444 s	

Figura C.9: Desempeño temporal de *CalculateMAP_v1*

Podemos observar que el tiempo total dedicado a 2325 llamadas de *CalculateMAP_v2* es de 0.27 segundos, lo que da un tiempo de ejecución unitario de $116\mu s$ mientras que el tiempo dedicado a 84 llamadas de *CalculateMAP_v1* es de 0.032 segundos, dando un tiempo de ejecución unitario de $381\mu s$.

El código de la función implementada se detalla a continuación.

```
1  function [MAPdisc] = CalculateMAP_v2(LongRes,Incremento,posx,posy,postheta,RAD)
2  %calcula el mapa discreto en forma de matriz a partir de la posición y el vector de distancias
       del telémetro. Se le debe indicar la longitud del recinto y la precisión de la
       discretización
3
4      AngInicial = 225;                          %parámetros del telémetro
5      Paso = 0.5;                                %valor positivo
6      AngFinal = -45;
7
8
9      res = fix(LongRes / Incremento);           %número de celdas en cada dimensión de la
           rejilla
10
11     MAPdisc = zeros(res,res);                  %se define el mapa discreto
12
13     NumMedidas = abs((AngInicial - AngFinal)/Paso) + 1;
14
15     if AngInicial > AngFinal                   %corrige signo del paso
16         Paso = -Paso;
```

```
17      end
18
19      if length(RAD) ~= NumMedidas
20          error('dimensiones incorrectas del vector de medidas');
21      end
22
23      ANG = AngInicial : Paso : AngFinal;         %vector de angulos
24
25
26      for i = 1 : NumMedidas              %bucle de identificación de coordenadas y discretización
27
28          if not(isnan(RAD(i)))                   %solo para impactos no nulos
29              x = posx + RAD(i)*cos(postheta + (ANG(i)*pi/180));
30              y = posy + RAD(i)*sin(postheta + (ANG(i)*pi/180));
31              if (x >= 0 && x < LongRes && y >= 0 && y < LongRes)  %solo entra si el objeto está
                    dentro del recinto
32                  xdisc = fix(x/Incremento) + 1;  %Se ubica la celda
33                  ydisc = fix(y/Incremento) + 1;
34                  MAPdisc(ydisc,xdisc) = 1;       %Se fuerza su valor a 1
35              end
36          end
37      end
38  end
```

C.7. CalculateMAP_v3

El objetivo de esta función es plasmar los datos provenientes del sensor en una *Occupancy Grid probabilística* realizando incrementos y decrementos en las celdas correspondientes.

Recordemos que una *Occupancy Grid probabilística* es, al igual que su predecesora booleana, una rejilla de posiciones discretizadas donde cada celda puede tomar un valor correspondiente a la probabilidad de que esté ocupada por un obstáculo. Así, deberemos determinar que acción tomar sobre las probabilidades de una celda en función de la información recibida, ya sea aumentándola o disminuyéndola en cierta medida.

Cabe destacar que esta función no define la *Occupancy Grid* sino que recibe la correspondiente al mapa anterior y realiza sobre esta los incrementos y/o decrementos correspondientes. Estos serán, al igual que los valores de las celdas, de tipo entero para facilitar los cálculos. Nótese que una *Occupancy Grid probabilística* correctamente definida debe tener sus valores acotados en un margen para poder considerarse probabilidades, pero esto no se implementa dentro de esta función para dotarla de una mayor compatibilidad, siendo la función de mapeo quien definirá la *Occupancy Grid*, decidirá su rango de valores y acotará los resultados de esta función.

Los incrementos y decrementos contemplados para los distintos casos son:

1. En caso de recibir un impacto positivo, se incrementa el valor de la celda correspondiente al obstáculo detectado según el valor del parámetro *incObjeto*(+2 por defecto).

2. En caso de recibir un impacto positivo, se decrementa el valor de todas las celdas correspondientes al recorrido del haz de luz hasta llegar al obstáculo detectado según el valor del parámetro *decAnteObjeto*(-3 por defecto). Esto da cuenta de que si se detecta un obstáculo, el espacio entre el robot y este se encuentra libre de obstáculos.[2]

3. En caso de recibir un impacto nulo, es decir, que no se detecta obstáculo en esa dirección, se decrementa el valor de todas las celdas correspondientes al recorrido del haz de luz hasta llegar al borde del recinto según el valor del parámetro *decViaLibre*(-2 por defecto), ya que se entiende que esas celdas se encuentran desocupadas.[3]

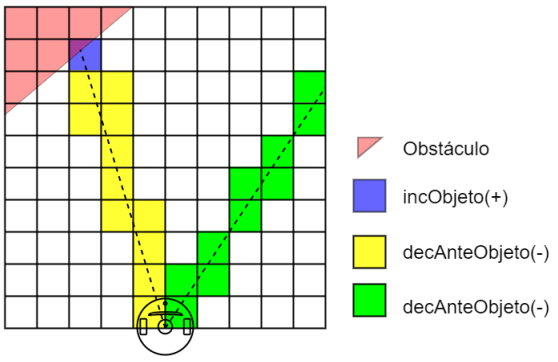


Figura C.10: Diagrama de los incrementos y decrementos para la *Occupancy Grid probabilística*

[2]Se considera sólo las celdas correspondientes a una línea de área nula ya que en teoría, el haz de luz tiene dimensiones despreciables.

[3]También se considera sólo una línea de área nula.

El procedimiento seguido por la función implementada es:

1. Se definen los parámetros del telémetro así como los valores de los incrementos y decrementos.

2. Con ayuda de un bucle, se analiza cada impacto. Para cada uno de estos, se define una matriz de ceros "máscara"de dimensiones similares al mapa que contendrá los incrementos y/o decrementos.

3. En caso de encontrar un impacto nulo:

   a) Con ayuda de la función *linspace* se definen dos vector de coordenadas correspondientes a la línea que sale del robot en la dirección actual. Por simplicidad, se ha tomado la longitud de esta línea como la diagonal máxima del recinto, por lo que en la mayoría de casos, sobrepasa el mismo. El número de puntos es el adecuado para que cada punto se corresponda a una celda.

   b) Se limita la línea al recinto, eliminando la porción de los vectores que sobrepasa este.[4]

   c) Se discretizan los puntos de la línea y con los índices obtenidos, se cambia el valor de las celdas de la máscara por el correspondiente al parámetro *DecViaLibre*.

4. En caso de encontrarse un impacto positivo:

   a) Se calculan las coordenadas del obstáculo mediante el método introducido en *CalculateMAP_v2*.

   b) Con ayuda de la función *linspace* se definen dos vectores de coordenadas correspondientes a la línea que une el robot con el obstáculo. El número de puntos es el adecuado para que cada punto se corresponda a una celda.

   c) En caso de detectar que la línea sobrepasa el recinto (en caso de haber un obstáculo por fuera), se limita a este desechando la parte de los vectores que sobrepasa.

   d) Se discretizan los puntos de la línea y con los índices obtenidos, se cambia el valor de las celdas de la máscara por el correspondiente al parámetro *DecAnteObjeto*.[5]

   e) Se discretizan las coordenadas del obstáculo y se cambia el valor de la celda por el correspondiente al parámetro *incObjeto*.

5. Aplica la máscara al mapa antiguo recibido mediante una suma. Esto se hace para cada iteración del bucle, realizando así los incrementos y/o decrementos correspondientes a cada impacto particular.

6. Por último, y ya fuera del bucle, se vuelca el valor del mapa antiguo sobreescrito en la variable correspondiente al mapa nuevo, que será la salida de nuestra función.

Los resultados de esta función se puede consultar en el apartado 3.3, en el subapartado *Entorno de simulación*. El código se detalla a continuación.

[4]Nótese que de estar el robot fuera del recinto, se considerará todo el vector como no válido y no se realizará decremento ninguno.

[5]Con este método se escribe también la celda del obstáculo por ser el último punto de la línea, pero no afecta al resultado final ya que luego sobreescribiremos en dicha celda el valor correspondiente.

```
function [MAPnew] = CalculateMAP_v3(MAPold,LongRes,Incremento,posx,posy,postheta,RAD)
%calcula el mapa discreto probabilístico en forma de matriz a partir de la posición y el
    vector de distancias del telémetro. Se le debe indicar la longitud del recinto y la
    precisión de la discretización


    AngInicial = 225;
    Paso = 0.5;                 %valor positivo
    AngFinal = -45;

    decViaLibre  = -2;                  %peso de encontrar un ángulo con Via Libre
    incObjeto    = +2;                  %peso de encontrar un objeto
    decAnteObjeto= -3;                  %peso de las posiciones entre el robot y un objeto
        encontrado

    res = fix(LongRes / Incremento);

    NumMedidas = abs((AngInicial - AngFinal)/Paso) + 1;

    if AngInicial > AngFinal    %corrige signo del paso
        Paso = -Paso;
    end

    if length(RAD) ~= NumMedidas
        error('dimensiones incorrectas del vector de medidas');
    end

    ANG = AngInicial : Paso : AngFinal;         %vector de angulos
    LongMax = LongRes * sqrt(2);        %parámetros de la línea en caso de impacto nulo
    numPuntosMax = LongMax / Incremento;

    for i = 1 : NumMedidas      %bucle de identificación de coordenadas y discretizción

        angreal = postheta + (ANG(i)*pi/180);
        MAPmask = zeros(res,res);       %máscara de incrementos/decrementos

        %Caso en que la medida particular no devuelva ningún resultado (via libre)
        if isnan(RAD(i))

            %crea línea en esa dirección de longitud máxima
            xlinepts = linspace(posx,(posx+LongMax*cos(angreal)),numPuntosMax);
            ylinepts = linspace(posy,(posy+LongMax*sin(angreal)),numPuntosMax);

            for j = 1:length(xlinepts)          %limita la línea al recinto
                if (xlinepts(j) < 0) || (xlinepts(j) >= LongRes) || (ylinepts(j) < 0) || (
                    ylinepts(j) >= LongRes)
                    xlinepts = xlinepts(1:(j-1));
                    ylinepts = ylinepts(1:(j-1));
                    break;
                end
            end

            %convierte las coordenadas en indices
            lineindex = sub2ind(size(MAPmask),(fix(ylinepts/Incremento)+1),(fix(xlinepts/
                Incremento)+1));
            MAPmask(lineindex) = decViaLibre;   %aplica los indices de la línea a la máscara

        %Caso en que se encuentra un objeto
        else
            x = posx + RAD(i)*cos(angreal);     %Calcula la posición del objeto
            y = posy + RAD(i)*sin(angreal);

            numpuntos = sqrt((posx-x)^2+(posy-y)^2) / Incremento;

            %Crea la linea desde el robot hasta el objeto
            xlinepts = linspace(posx,x,numpuntos);
            ylinepts = linspace(posy,y,numpuntos);

            %limita la línea al recinto
            if (min(xlinepts) < 0 || (max(xlinepts) >= LongRes || min(ylinepts) < 0) || max(
                ylinepts) >= LongRes)
                for j = 1:length(xlinepts)
```

```
67                      if (xlinepts(j) < 0) || (xlinepts(j) >= LongRes) || (ylinepts(j) < 0) || (
                            ylinepts(j) >= LongRes)
68                          xlinepts = xlinepts(1:(j-1));
69                          ylinepts = ylinepts(1:(j-1));
70                          break;
71                      end
72                  end
73              end
74
75              %convierte las coordenadas en indices
76              lineindex = sub2ind(size(MAPmask),(fix(ylinepts/Incremento)+1),(fix(xlinepts/
                    Incremento)+1));
77              MAPmask(lineindex) = decAnteObjeto;  %Aplica los indices de la línea a la máscara
78
79              %solo entra si el objeto está dentro del recinto
80              if (x >= 0 && x < LongRes && y >= 0 && y < LongRes)
81                  xdisc = fix(x/Incremento) + 1;  %Calcula posición discretizada del objeto
82                  ydisc = fix(y/Incremento) + 1;
83                  MAPmask(ydisc,xdisc) = incObjeto;        %Aplica máscara de objeto
84              end
85
86          end
87
88              MAPold = MAPold + MAPmask;                          %Aplica la máscara
89      end
90
91          MAPnew = MAPold;                                    %Vuelca la solución para su salida
92
93  end
```

Apéndice D

Pruebas Complementarias

D.1. Mapping_v5: Bucle doble VS media

Se estudian dos métodos que se propusieron para realizar la tarea de comparar dos matrices de iguales dimensiones. El objetivo es obtener un número proporcional al número de elementos iguales a '1' que tienen en común. Aclarar que los elementos de la matriz son de tipo lógico, es decir que sólo pueden ser '0' o '1'.

El primer método que se propuso trata de, con la ayuda de un bucle doble, barrer todas las posiciones de la matriz y para cada una de ellas, si ambos elementos son 1, se incrementa una variable de cuenta. Para poder controlar el peso computacional de este algoritmo, se ha definido un parámetro de precisión (*matchpres*) que corresponde al incremento de posición que se hace durante el barrido. Por ejemplo, si *matchpres* = 1, se comprobaran todos los elementos de la matriz y si *matchpres* = 3, se comprobara uno de cada 3 elementos por fila y uno de cada 3 por columna.

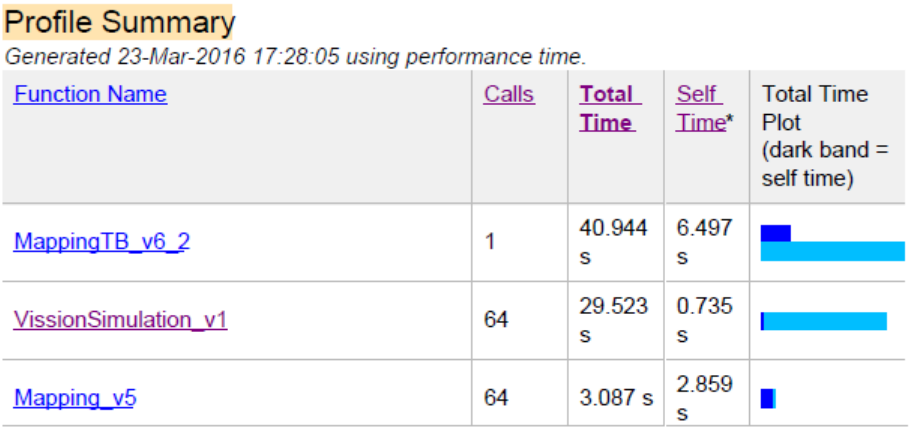
Matlab tiene la opción de analizar la respuesta temporal de una función, el resultado de analizar el entorno de simulación correspondiente se muestra a continuación. Recordar que los datos que nos interesan son los correspondientes a *Mapping_v5*.

Profile Summary

Generated 23-Mar-2016 17:00:01 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
MappingTB_v6_2	1	48.659 s	6.482 s	
VissionSimulation_v1	64	29.208 s	0.718 s	
Mapping_v5	64	11.217 s	10.985 s	

Figura D.1: Desempeño temporal de *Mapping_v5* con bucle de precisión 1



Profile Summary

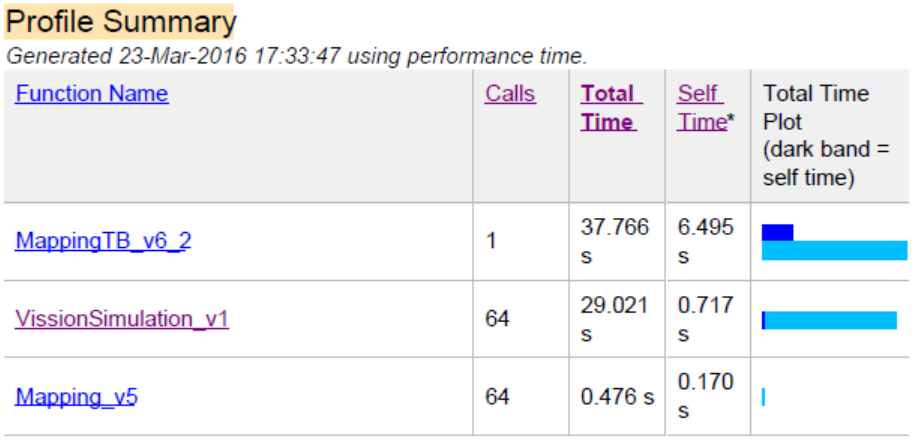
Generated 23-Mar-2016 17:28:05 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
MappingTB_v6_2	1	40.944 s	6.497 s	
VissionSimulation_v1	64	29.523 s	0.735 s	
Mapping_v5	64	3.087 s	2.859 s	

Figura D.2: Desempeño temporal de *Mapping_v5* con bucle de precisión 2

Observamos que los tiempos totales dedicados a *Mapping_v5* son de 11.217 y 3.087 segundos respectivamente. Si quisiéramos conocer el tiempo de ejecución de esta función, habría que dividir este número por el número de llamadas que tuvo. Pero ya que vamos a analizar el mismo entorno de simulación, donde el número de llamadas a la función es siempre el mismo, vamos a analizar directamente el tiempo total dedicado a la función.

El segundo método propuesto hace uso del operador AND para obtener una tercera matriz donde haya 1's en las posiciones que ambas matrices operadas tenían un '1'. A continuación se halla la media de los elementos de la matriz con ayuda de la función *mean2*, que será proporcional a la cantidad de 1's. Los resultados temporales del entorno de simulación aplicando este método se muestran a continuación.



Profile Summary

Generated 23-Mar-2016 17:33:47 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
MappingTB_v6_2	1	37.766 s	6.495 s	
VissionSimulation_v1	64	29.021 s	0.717 s	
Mapping_v5	64	0.476 s	0.170 s	

Figura D.3: Desempeño temporal de *Mapping_v5* con media de AND

Como podemos observar, el tiempo total dedicado a la función es de 0.476 segundos aplicando este método, superando considerablemente los resultados obtenidos con el anterior método aun con *matchpres* = *2*. Por lo tanto queda demostrado que este método es mucho más eficiente y es el que se aplica en *Mapping_v5*, el otro método se deja comentado dentro del código, que se puede consultar en el apéndice B.

Bibliografía

[1] https://es.mathworks.com/matlabcentral/fileexchange/48307-print-table.

[2] AVOTS, D., LIM, E., R.THIBAUX, AND THRUN, S. A probabilistic technique for simultaneous localization and door state estimation with mobile robots in dynamic environments. *Submited por publication* (2002).

[3] BARRIENTOS, A. *Fundamentos de Robotica*. Mc Graw-Hill, 2007.

[4] BATURONE, A. O. *Robotica: manipuladores y robots moviles*. Marcombo, 1991.

[5] BISWAS, R., LIMKETKAI, B., S.SANNER, AND THRUN, S. Towards object mapping in dynamic environments with mobile robots. *Submited por publication* (2002).

[6] BURGARD, W., FOX, D., JANS, H., MATENAR, C., AND THRUN, S. Sonar-based mapping of large-scale mobile robot environments using em. *In Proceedings of the International Conference on Machine Learning, Bled, Slovenia* (1999).

[7] CSORBA, M. *Simultaneous Localisation and Map Building*. PhD thesis, University oj Oxford, 1997.

[8] DISSANAYAKE, G., DURRANT-WHYTE, H., AND BAILEY, T. A computationally efficient solution to the simultaneous localisation and map building (slam) problem. *Working notes of ICRA2000 Workshop W4: Mobile Robot Navigation and Mapping* (2000).

[9] DISSANAYAKE, G., NEWMAN, P., CLARK, S., DURRANT-WHYTE, H., AND CSORBA, M. A solution to the simultaneous localisation and map building (slam) problem. *IEEE Transactions of Robotics and Automation* (2001).

[10] ELFES, A. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation* (1987).

[11] ENGELSON, S., AND MCDERMOTT, D. Error correction in mobile robot map learning. *In Proceedings of the 1992 IEEE International Conference on Robotics and Automation, Nice, France* (1992).

[12] HAKIM, S., AND BOULANGER, P. Sensor based creation of indoor virtual environment models. *In Proc. of the 4th Internetional Conference on Virtual Systems and Multimedia (VSMM)* (1997).

[13] LEONARD, J., DURRANT-WHYTE, H., AND COX, I. Dynamic map building for an autonomous mobile robot. *International Journal of Robotics Research* (1992).

[14] MONTEMERLO, M., THRUN, S., KOLLER, D., AND WEGBREIT, B. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Submited por publication* (2002).

[15] NEWMAN, P. *On the Structure and Solution of the Simultaneous Localisation and Map Building Problem*. PhD thesis, Australian Centre for Field Robotics, University of Sydney, Sydney, Australia, 2000.

[16] PIERCE, D., AND KUIPERS, B. Learning to explore and build maps. *In Proceedings of the Twelfth National Conference on Artificial Intelligence, Menlo Park* (1994).

[17] SHATKAY, H., AND KAELBLING, L. Learning topological maps with weak local odometric information. *In Proceedings of IJCAI-97. IJCAI, Inc.* (1997).

[18] SIMMONS, R., APFELBAUM, D., BURGARD, W., FOX, M., AN MOORS, D., THRUN, S., AND YOUNES, H. Coordination for multi-robot exploration and mapping. *In Proceedings of the AAAI National Conference on Artificial Intelligence, Austin, TX* (2000).

[19] THRUN, S. Exploration and model building in mobile robot domains. *In E. Ruspini, editor, Proceedings of the IEEE International Conference on Neural Networks, San Francisco, CA* (1993).

[20] THRUN, S. Learning occupancy grids with forward models. *In Proceedings of the Conference on Intelligent Robots and Systems (IROS 2001), Hawaii* (2001).

[21] THRUN, S. A probabilistic online mapping algorithm for teams of mobile robots. *International Journal of Robotics Research* (2001).

[22] THRUN, S. Robotic mapping: A survey. *Carnegie Mellon University* (2002).

[23] THRUN, S., BURGARD, W., AND D.FOX. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. MIT Press, 2005.

[24] THRUN, S., BURGARD, W., AND FOX, D. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping. *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), San Francisco, CA* (2000).

[25] THRUN, S., FOX, D., AND BURGARD, W. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning* (1998).