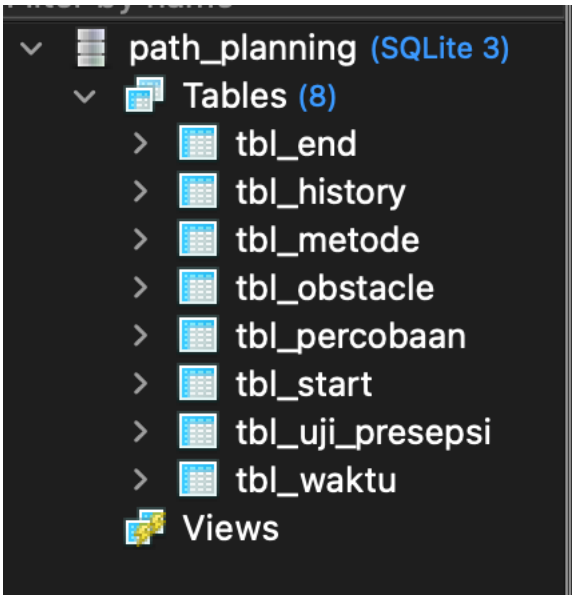
BAB V. IMPLEMENTASI DAN PENGUJIAN

5.1 Implementasi

Setelah dilakukan perancangan sistem, maka selanjutnya adalah implementasi sistem sesuai dengan perancangan yang dilakukan. Pada bagian ini menjelaskan tentang hasil dari sistem yang telah dibangun. Implementasi dijelaskan secara detail secara visual dengan tampilan gambar dan potongan kode program atau *listing code*, sebagai berikut:

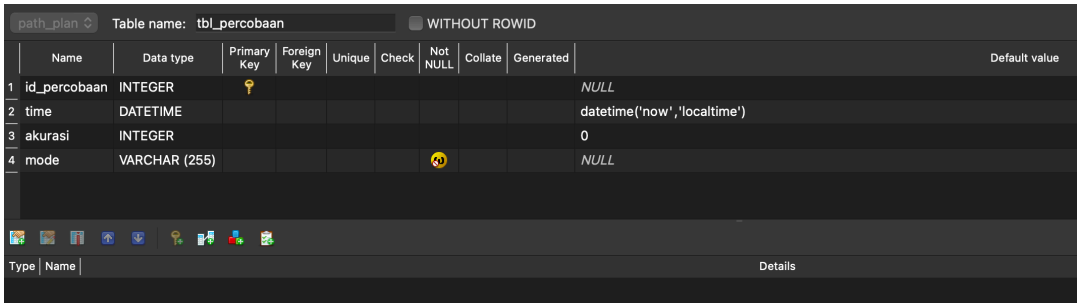
5.1.1 Implementasi *Database*

Implementasi *database* sesuai dengan rancangan yang telah dilakukan dengan menggunakan *database SQLite3* yang digunakan untuk menyimpan data dalam sistem, Gambar 5.1 merepresentasikan implementasi *database* sistem.



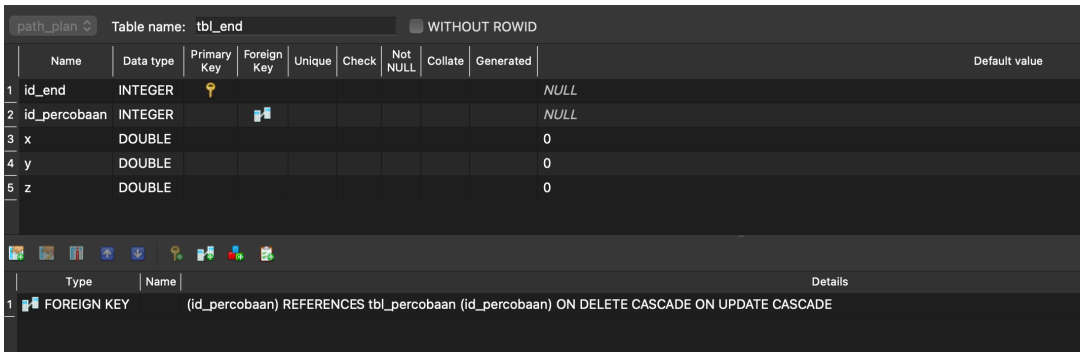
Gambar 5.1 Implementasi Tabel *Database* Sistem

Tabel – tabel yang diimplementasikan ke sistem berjumlah delapan tabel antara lain, tbl_end, tbl_history, tbl_metode, tbl_obstacle, tbl_percobaan, tbl_start, tbl_uji_presepsi, dan tbl_waktu. Gambar 5.2 merupakan representasi dari tabel percobaan yang digunakan untuk menyimpan setiap percobaan yang dilakukan oleh sistem terhadap perencanaan rute pencarian menggunakan algoritma *Improved* A*. Dalam tbl_percobaan memiliki beberapa atribut antara lain id_percobaan (*primary key*), time, akurasi dan mode.



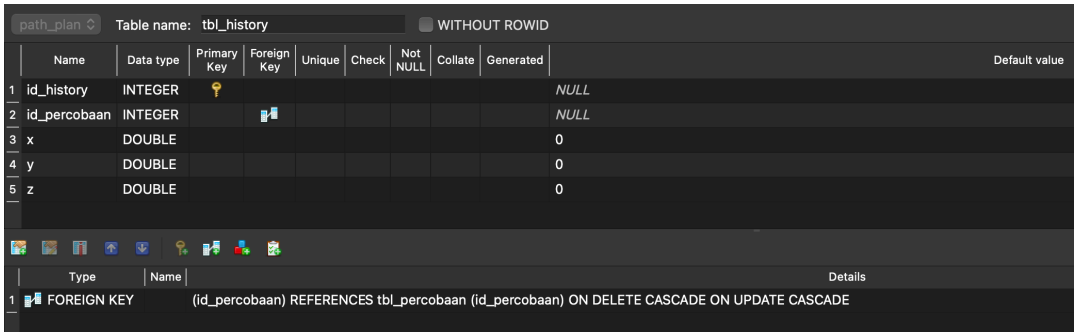
Gambar 5.2 Implementasi Tabel Percobaan

Gambar 5.3 merupakan representasi dari tabel *end* atau titik tujuan akhir robot yang digunakan untuk menyimpan setiap titik akhir dari tujuan yang diinginkan dalam setiap percobaan. Dalam tbl_end memiliki beberapa atribut antara lain id_history (*primary key*), id_percobaan (*foreign key*) dari tabel percobaan pada atribut id_percobaan, x menyatakan koordinat x, y menyatakan koordinat y dan z menyatakan posisi koordinat z atau arah hadap dari robot.



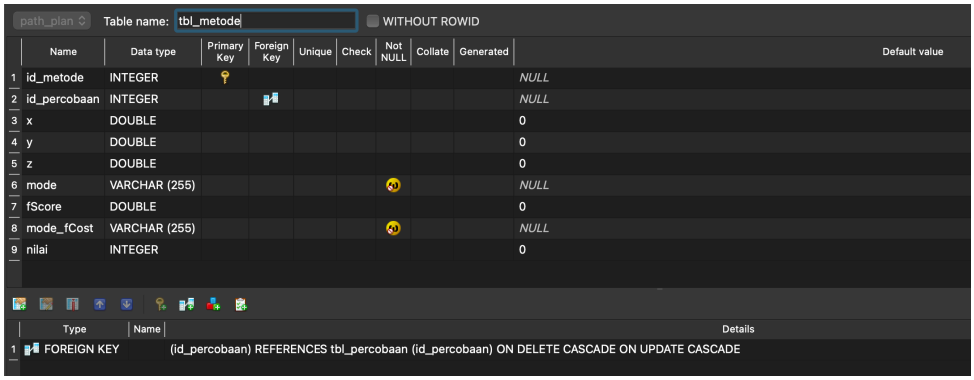
Gambar 5.3 Implementasi Tabel *End*

Gambar 5.4 merupakan representasi dari tabel *history* atau tabel perpindahan posisi koordinat robot dari titik awal yang ditentukan menuju titik akhir. Dalam tbl_history memiliki beberapa atribut antara lain id_history (*primary key*), id_percobaan (*foreign key*) dari tabel percobaan pada atribut id_percobaan, x menyatakan koordinat x, y menyatakan koordinat y dan z menyatakan posisi koordinat z atau arah hadap dari robot.



Gambar 5.4 Implementasi Tabel *History*

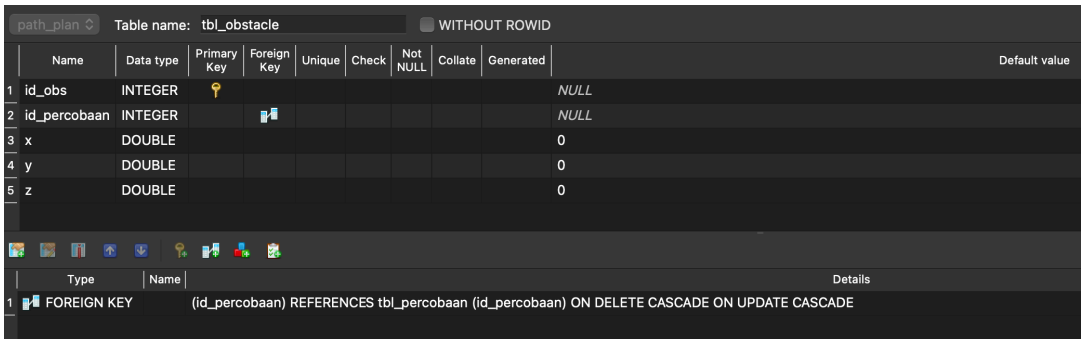
Gambar 5.5 merupakan representasi dari tabel metode yang digunakan untuk menyimpan *node* atau pemilihan jalur yang dilewati pada perencanaan rute menggunakan algoritma A* dan *Improved* A*. Dalam tbl_metode memiliki beberapa atribut antara lain id_metode (*primary key*), id_percobaan (*foreign key*) dari tabel percobaan pada atribut id_percobaan, x menyatakan koordinat x, y menyatakan koordinat y, z menyatakan posisi koordinat z atau arah hadap dari robot, model menyatakan pemilihan terhadap uji coba secara simulasi menggunakan robot *dummy* atau menggunakan robot hardware, fScore menyatakan kalkulasi fungsi biaya yang telah dilewati pada setiap *node* yang direpresentasikan pada persamaan 2.1, mode_fCost menyatakan pemilihan fungsi biaya yang digunakan yaitu *Manhattan Distance* atau *Euclidean Distance*, dan nilai menyatakan representasi apakah *node* bersangkutan dianggap bernilai efisien dengan representasi nilai 1 atau tidak efisien dengan representasi nilai 0.



Gambar 5.5 Implementasi Tabel Metode

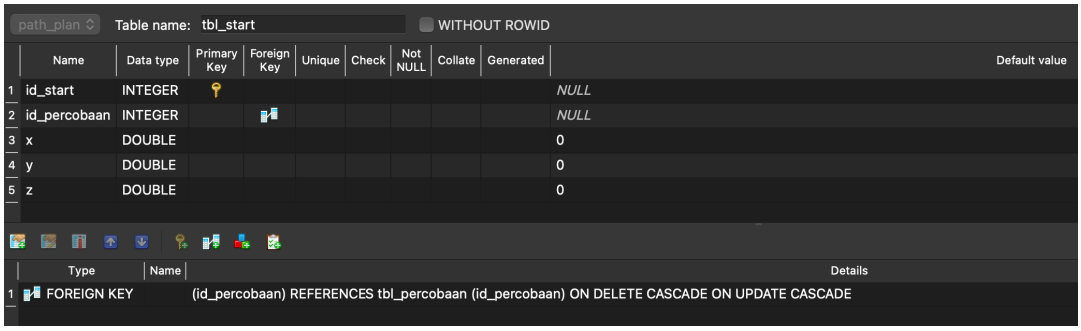
Gambar 5.6 merupakan representasi dari tabel *obstacle* atau robot halangan pada setiap percobaan yang dilakukan. Dalam tbl_history memiliki beberapa atribut antara lain id_obstacle (*primary key*), id_percobaan (*foreign key*) dari tabel

percobaan pada atribut id_percobaan, x menyatakan koordinat x, y menyatakan koordinat y dan z menyatakan posisi koordinat z atau arah hadap dari robot.



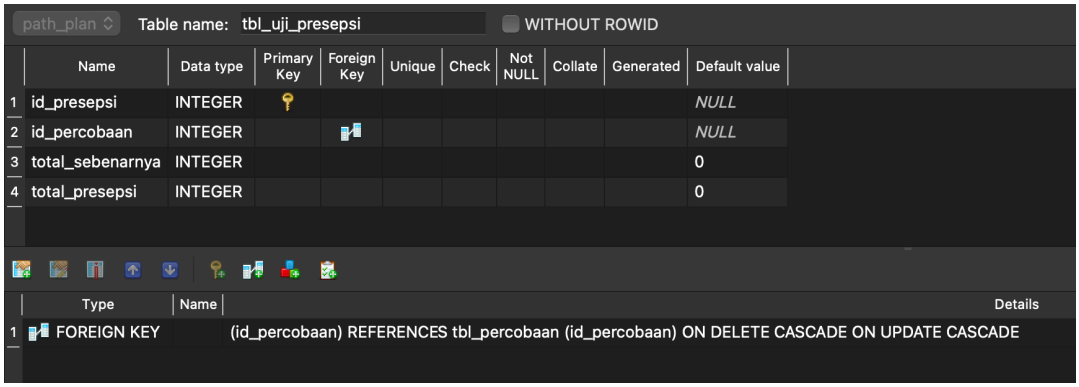
Gambar 5.6 Implementasi Tabel *Obstacle*

Gambar 5.7 merupakan representasi dari tabel *start* atau titik awal robot yang digunakan untuk menyimpan setiap titik koordinat robot saat ini dalam setiap percobaan. Dalam tbl_start memiliki beberapa atribut antara lain id_start (*primary key*), id_percobaan (*foreign key*) dari tabel percobaan pada atribut id_percobaan, x menyatakan koordinat x, y menyatakan koordinat y dan z menyatakan posisi koordinat z atau arah hadap dari robot.



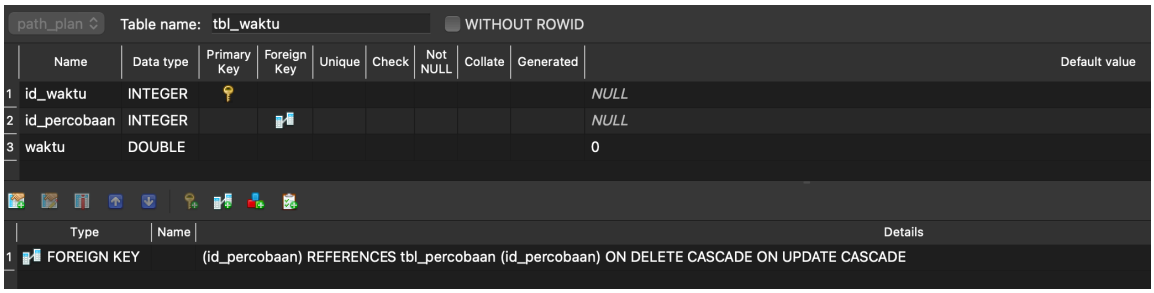
Gambar 5.7 Implementasi Tabel *Start*

Gambar 5.8 merupakan representasi dari tabel uji persepsi yang digunakan untuk melakukan perhitungan terhadap akurasi persepsi manusia dengan perencanaan rute yang dilakukan oleh algoritma *Improved* A* pada setiap percobaan yang dilakukan. Dalam tbl_uji_presepsi memiliki beberapa atribut antara lain id_presepsi (*primary key*), id_percobaan (*foreign key*) dari tabel percobaan pada atribut id_percobaan, total_sebenarnya menyatakan total sebenarnya yang dihasilkan oleh algoritma *Improved* A* pada setiap percobaan, total_presepsi menyatakan hasil pengujian dari persepsi manusia mengenai total yang harus dilewati oleh algoritma *Improved* A* dalam setiap percobaan.



Gambar 5.8 Implementasi Tabel Uji Persepsi

Gambar 5.9 merupakan representasi dari tabel waktu yang digunakan untuk menyatakan kalkulasi waktu dalam menyelesaikan perencanaan rute dengan menggunakan algoritma *Improved* A*. Dalam tbl_waktu memiliki beberapa atribut antara lain id_waktu (*primary key*), id_percobaan (*foreign key*) dari tabel percobaan pada atribut id_percobaan, dan waktu yang menyatakan waktu yang dicapai untuk penyelesain metode dengan menggunakan fungsi jarak *Manhattan Distance* atau *Euclidean Distance.*



Gambar 5.9 Implementasi Tabel Waktu

5.1.2 Implementasi Kode Program

Implementasi beberapa potongan kode program alur proses kerja yang ada dalam sistem sesuai dengan rancangan yang telah dilakukan sebelumnya. Sistem dibuat menggunakan bahasa pemrograman *Python 3.x*, QtDesigner, dan PyQt5. Sistem yang dibuat pada penelitian ini berbasis aplikasi *desktop*. Implementasi kode program berdasarkan proses analisis dan perancangan dijelaskan secara detail, sebagai berikut:

A. *Socket Programming* UDP

Socket Programming UDP digunakan untuk mengambil data posisi koordinat robot secara *real time* dengan menggunakan IP Address dan *port* yang telah ditentukan. Data posisi koordinat bersangkutan menjadi acuan untuk perencanaan rute dengan menggunakan algoritma *Improved* A*. Berikut ini merupakan *source code* kode program *socket udp* untuk pengiriman dan pengambilan data pada robot yang sudah ditransformasikan terhadap peta statis yang dibuat:

```
path_hardware = QtCore.pyqtSignal(object)

@QtCore.pyqtSlot()
def run(self):
     self.server_udp = socket.socket(socket.AF_INET,
     socket.SOCK_DGRAM, socket.IPPROTO_UDP)
     self.port = 8000
     self.my_ip = []
     self.address = []
     while(classKirim.isLoopUDP == 1):
          if(classKirim.isActiveSimulasi == 1):
               if(self.clientAdd == 0):
                    self.server_udp.setsockopt(
                    socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
                    self.server_udp.setsockopt(
                    socket.SOL_SOCKET, socket.SO_REUSEPORT, 1)
                    self.server_udp.setsockopt(
                    socket.SOL_SOCKET,.SO_BROADCAST, 1)
                    self.server_udp.bind(("", self.port))
                    self.clientAdd = 1
                    self.my_ip = socket.gethostbyname_ex(
                    socket.gethostname())[-1]
               else:
                    data, address = self.server_udp.recvfrom(1024)
                    temp_receive = pickle.loads(data)
                    if(address[0] not in self.my_ip):
                         ext = temp_receive["exit"]
                         if(ext == 1):
                              indeks = self.address.index(
                              address[0])
                              self.address.pop(indeks)
                              self.pos = len(self.address)
                              self.hapus_client.emit(indeks)
                         else:
                              x = temp_receive["pos"][0]
                              y = temp_receive["pos"][1]
                              z = temp_receive["pos"][2]
                              x_transformation = (classKirim.maxX/2)- x
                              y_transformation= (classKirim.maxY/2)-
                             (y * -1)
                              self.path_hardware.emit(temp_receive)
```

B. Manajemen *Create* Data

Dalam manajemen data pada sistem perencanaan rute *Improved* A*, disediakan fungsi yang digunakan untuk menambahkan data pada implementasi database yang dibuat pada perancangan sistem. Berikut ini merupakan fungsi yang digunakan untuk menambahkan data ke dalam tabel sesuai dengan parameter yang dipanggil pada fungsi "**insert_data**".

```
def insert_data(self, table_name, data):
    if(table_name == "tbl_percobaan"):
         query = """INSERT INTO "{0}"(mode) VALUES
         ('{1}')""".format(table_name, data)
         self.cursor.execute(query)
         self.mydb.commit()
         self.last_id = self.cursor.lastrowid

     elif(table_name == "tbl_metode"):
         mode = ["Astar Search", "Improved Astar"]
         for counter in range(len(data)):
         if(counter == 1):
              query = """INSERT INTO tbl_uji_presepsi
              (id_percobaan,total_sebenarnya) VALUES
              ({0},{1})""".format(.last_id, len(data[counter]))
              self.cursor.execute(query)
              timer = round(classKirim.timer[0], 5)
              query = """INSERT INTO tbl_waktu
         (id_percobaan,waktu) VALUES ({0},{1})""".format(
              self.last_id, timer)
              self.cursor.execute(query)
              items = data[counter]
              for path in items:
                   x = path[0]
                   y = path[1]
                   fScore = ui.astarM.fScore.get(path, 0)
                   insert_data = [(x, y, fScore)]
                   query = """INSERT INTO
                   "{0}"(id_percobaan,x,y,fScore,mode,mode_fCost)
                   VALUES ({1},?,?,?,'{2}','{3}')""".format(
                   table_name, self.last_id, mode[counter],
                   ui.mode_fcost)
                   self.cursor.executemany(query, insert_data)
                   self.mydb.commit()

     elif(table_name == "tbl_start" or table_name == "tbl_end" or
     table_name == "tbl_obstacle"):
         query = """INSERT INTO "{0}"(id_percobaan,x,y) VALUES
         ({1},?,?)""".format(table_name, self.last_id)
         self.cursor.executemany(query, data)
         self.mydb.commit()

     elif(table_name == "tbl_history"):
         query = """INSERT INTO "{0}"(id_percobaan,x,y,z) VALUES
         ({1},?,?,?)""".format(table_name, self.last_id)
          if(classKirim.isActiveSimulasi == 0):
              self.cursor.executemany(query, data)
              self.mydb.commit()
          else:
              try:
                   self.cursor_history.executemany(query, data)
```

```
                    self.mydb.commit()
               except:
                    pass
```

C. Manajemen *Read* Data

Dalam manajemen data pada sistem perencanaan rute *Improved* A*, disediakan fungsi yang digunakan untuk mengambil data yang ada pada database sistem yang telah dibuat. Berikut ini merupakan fungsi yang digunakan untuk menampilkan data sesuai dengan parameter yang dipanggil pada fungsi yang dideklarasikan.

- Fungsi "**get_data_percobaan**" menyatakan pengambilan data pada tabel percobaan yang berfungsi untuk mengambil data detail percobaan perencanaan rute *Improved* A* pada aplikasi. Berikut merupakan representasi dari *source code* program.

```
def get_data_percobaan(self):
     try:
          query = """SELECT * FROM tbl_percobaan ORDER BY
          id_percobaan ASC"""
          result = self.cursor.execute(query)
          self.mydb.commit()
          result = result.fetchall()
          return result
     except sqlite3.Error as e:
          print("Data View Percobaan Error: ", e)
```

- Fungsi "**get_query**" menyatakan pengambilan data pada tabel percobaan yang berfungsi untuk mengambil data detail percobaan perencanaan rute *Improved* A* pada aplikasi. Berikut merupakan representasi dari *source code* program.

```
def get_query(self, table_name, id):
     try:
          query = """SELECT * FROM "{0}" WHERE id_percobaan =
          {1}""".format(table_name, id)
          data = self.cursor.execute(query)
          data = data.fetchall()
          self.mydb.commit()
          return data
     except sqlite3.Error as e:
          print(e)
```

- Fungsi "**get_query_and**" menyatakan pengambilan data pada tabel sesuai dengan parameter yang dipanggil pada fungsi bersangkutan dengan kondisi *query* "**AND**". Berikut merupakan representasi dari fungsi *source code* program.

```
def get_query_and(self, table_name, id, cond):
     try:
          query = """SELECT * FROM "{0}" WHERE id_percobaan = {1}
          AND {2}""".format(table_name, id, cond)
```

```
            data = self.cursor.execute(query)
            data = data.fetchall()
            self.mydb.commit()
            return data
        except sqlite3.Error as e:
            print(e)
```

- Fungsi "**get_manual_query**" menyatakan pengambilan data pada tabel sesuai dengan parameter yang dipanggil pada fungsi bersangkutan. Parameter fungsi bersangkutan bernilai *query* yang dilempar terhadap fungsi bersangkutan, contoh *query* yang digunakan antara lain, yaitu "**INNER JOIN**", dll. Berikut merupakan representasi dari *source code* program.

```
def get_manual_query(self, query):
    try:
        data = self.cursor.execute(query)
        data = data.fetchall()
        self.mydb.commit()
        return data
    except Exception as e:
        print(e)
```

D. Manajemen *Delete* Data

Dalam manajemen data pada sistem perencanaan rute *Improved* A*, disediakan fungsi yang digunakan untuk menghapus data pada tabel sesuai dengan parameter yang dipanggil pada fungsi bersangkutan. Parameter "table_name" menyatakan tabel mana yang ingin dihapus, "where" menyatakan kondisi dari atribut pada tabel, dan "id" menyatakan id dari data yang ingin dihapus. Berikut merupakan representasi dari *source code* program.

```
def delete_data(self, table_name, where, id):
    try:
        query = """DELETE FROM "{0}" WHERE "{1}" = {2}
        """.format(table_name, where, id)
        self.cursor_delete.execute(query)
        self.mydb.commit()
        self.last_id = 0
    except sqlite3.Error as e:
        print("Delete Error: ", e)
```

5.1.3 Implementasi Proses Perhitungan Metode

Pada proses perhitungan metode akan dijelaskan bagaimana alur dari penerapan metode yang dilakukan pada penelitian ini. Implementasi dari penerapan metode akan dilampirkan dalam bentuk *source code* program. Berikut merupakan langkah – langkah yang dilakukan pada proses algoritma yang dibuat pada sisi program.

A. Implementasi Kode Program *A-star Search* (A*)

- Menentukan arah pencarian *node* tetangga

Data arah pencarian *node* tetangga akan menjadi acuan terhadap eksplorasi *node* untuk menentukan antrian *node* dengan fungsi biaya kalkulasi paling minimum. Berikut merupakan representasi dari *source code* program.

```
self.dir_row = [-0.5, -0.5, 0, +0.5, +0.5, +0.5, 0, -0.5]
self.dir_col = [0, +0.5, +0.5, +0.5, 0, -0.5, -0.5, -0.5]
```

- Menentukan fungsi perhitungan *heuristic*

Fungsi perhitungan *heuristic* berfungsi untuk menghitung jarak fungsi biaya dari setiap node ketika melakukan eksplorasi *node* dengan fungsi biaya minimum terhadap *node* tetangga. Berikut merupakan representasi dari *source code* program.

```
def heuristic(self, start, end):
     start = np.array(start)
     end = np.array(end)
     res = 0.
     if(self.mode_fCost == 0):
          # Manhattan Distance
          res = np.sum(np.abs(end - start))
     elif(self.mode_fCost == 1):
          # Euclidean Distance
          res = np.sqrt(np.sum((end - start)**2))
     return res
```

- Menentukan variabel penampung A*

Menentukan variabel penampung yang digunakan pada metode A*. Variabel yang digunakan antara lain yaitu *openSet, closedSet, cameFrom, start, end*. *openSet* digunakan sebagai penampung *node* antrian terhadap *node* tetangga yang telah dieksplorasi, *closedSet* menyatakan *node* yang telah dieksplorasi, dimana untuk pencarian *node* yang baru harus memperhatikan bahwa ketika data *node* telah ada pada variabel *closedSet* maka data perhitungan kalkulasi biaya tidak dapat di perbarui, sedangkan *cameFrom* menyatakan penampung data *node* dengan nilai fungsi biaya minimum setiap kali melakukan fungsi pencarian terhadap *node* tetangga. Ouput dari *cameFrom* menyatakan output dari perencanaan rute algoritma A*, *start* menyatakan titik koordinat robot asli yang sudah dipetakan pada map statis, dan *end* menyatakan koordinat titik tujuan terakhir dari robot. Berikut merupakan representasi dari *source code* program.

```
@QtCore.pyqtSlot()
def run(self):
     last_time = time.time()
     self.path = []
     self.dir_row = [-0.5, -0.5, 0, +0.5, +0.5, +0.5, 0, -0.5]
     self.dir_col = [0, +0.5, +0.5, +0.5, 0, -0.5, -0.5, -0.5]
```

```
self.cameFrom = {}
self.startP = classKirim.start
self.endP = classKirim.end
self.gScore = {self.startP: 0}
self.fScore   =   {self.startP:   self.heuristic(self.startP,
self.endP)}
self.openSet = []
self.closedSet = set()
self.openSet.append((self.startP, self.fScore[self.startP])
self.isLoop = 0
```

- Proses pencarian rute dengan menggunakan Algoritma A*

Setelah dilakukan proses penentuan variabel pada program dengan menggunakan *list,* dan *set,* maka akan dilakukan proses pencarian rute dengan menggunakan algoritma A* dengan menghitung biaya paling minimum terhadap *node* yang dieksplorasi. Berikut merupakan representasi dari *source code* program.

```
fin = np.array(self.endP) - np.array(self.startP)
fin = np.linalg.norm(fin)
if(fin <= 1.5):
     self.isLoop = 1
     self.path = [self.endP]
     tentative_gscore = self.gScore[self.startP] + \
     self.heuristic(self.startP, self.endP)
     self.fScore[self.endP] = tentative_gscore + \
     self.heuristic(self.startP, self.endP)

     classKirim.timer[0] = time.time() - last_time
else:
     self.cond_break = False
     # Jarak Aman Terhadap Obstacle
     enorm_locking = 0.6375
     # Jarak Minimum Terhadap Titik Tujuan Akhir
     erfin = 0.75

     while(self.isLoop == 0):
          if(len(self.openSet) > 0):
               self.openSet.sort(reverse=False, key=lambda
               idx: idx[1])
               cur = self.openSet.pop(0)[0]
               fin = np.array(self.endP) - np.array(cur)
               fin = np.linalg.norm(fin)
               if(fin <= erfin):
                    self.cond_break = True

               if(self.cond_break == True):
                    self.path = []
                    last_neighbour = cur
                    # Proses Looping Path A*
                    while(cur in self.cameFrom):
                         self.path.append(cur)
                         cur = self.cameFrom[cur]

                    self.path = self.path[::-1]
                    last_path = self.path[-1]
                    enorm_finish = np.array(self.endP) -
                    np.array(last_path)
                    enorm_error = np.linalg.norm(enorm_finish)
```

```
                    if(enorm_error <= 0.1875):
                        self.path.pop(len(self.path) - 1)
                    self.path += [self.endP]
                    self.isLoop = 1
                    t_gscore = self.gScore[last_neighbour] +
                    self.heuristic(last_neighbour, self.endP)
                    self.fScore[self.endP]   =   t_   gscore   +
                    self.heuristic(last_neighbour, self.endP)

                else:
                    self.closedSet.add(cur)

                    for i in range(len(self.dir_col)):
                        rr = cur[0] + self.dir_row[i]
                        cc = cur[1] + self.dir_col[i]
                        neighbor = (rr, cc)

                        if(rr < 0 or cc < 0):
                            continue
                        if(rr   >   classKirim.maxX   or   cc   >
                        classKirim.maxY):
                            continue

                        # Perhitungan Jarak Terhadap Obstacle
                        conditional = False
                        obs_total = classKirim.obs

                        if(classKirim.isActiveSimulasi == 1):
                            obs_total = classKirim.obs +
                            classKirim.obsO

                        for obs in obs_total:
                            new_point = np.array(obs)
                            now_node = np.array(neighbor)
                            enorm = new_point - now_node
                            enorm = np.linalg.norm(enorm)
                            if(enorm <= enorm_locking):
                                conditional = True
                                break
                            if(conditional):
                                continue
                            tentative_gscore=self.gScore[cur]
                            + self.heuristic(cur, neighbor)

                            if(neighbor in self.closedSet and
                            tentative_gscore >=
                            self.gScore.get(neighbor, 0)):
                                continue

                            if(tentative_gscore <
                            self.gScore.get(neighbor, 0) or
                            neighbor not in [idx[0] for idx in
                            self.openSet]):
                                self.cameFrom[neighbor]=cur
                                self.gScore[neighbor] =
                            tentative_gscore
                                self.fScore[neighbor]        =
                                tentative_gscore +
                                self.heuristic(neighbor,
                                self.endP)
```

```
                                        self.openSet.append(
                                    (neighbor,
                                        self.fScore[neighbor]))
```

B. Implementasi Kode Program *Improved A-star Search* (A*)

Setelah dilakukan proses perencanaan rute dengan menggunakan A*, langkah selanjutnya yaitu menormalisasi rute perencanaan dengan menggunakan metode *Improved* A* dengan kombinasi *Digital Differential Analyzer* (DDA). Berikut langkah – langkah yang dilakukan pada proses algoritma A*, antara lain:

Menentukan variabel penampung *Improved* A*

Menentukan variabel penampung yang digunakan pada metode Improved A*. Variabel yang digunakan antara lain yaitu *start, end, last_path, inc, new_path, length. Start* digunakan untuk menyimpan data posisi koordinat robot saat ini, *end* digunakan untuk menyimpan data posisi koordinat tujuan akhir dari robot, *last_path* menyatakan posisi koordinat yang dihasilkan dari algoritma A*, *inc* digunakan untuk melakukan iterasi atau perulangan dari titik koordinat awal robot sampai titik koordinat akhir tujuan robot, *new_path* merupakan output dari perencanaan *Improved* A*, dan *length* merupakan total dari panjang rute perjalanan yang dihasilkan dari algoritma A*. Berikut merupakan representasi dari *source code* program *Improved* A*.

```
def __init__(self, path):
     # Jarak Aman Obstacle
     self.error_obs = 0.6375
     self.startP = classKirim.start
     self.endP = classKirim.end
     self.last_path = copy(path)
     self.inc = 0
     self.new_path = [self.startP]
     self.length = len(self.last_path)
     self.last_inc = 0
```

- Proses pencarian rute dengan menggunakan algoritma *Improved* A*.

Setelah dilakukan proses penentuan variabel pada program dengan menggunakan *list,* maka akan dilakukan proses pencarian rute dengan menggunakan algoritma *Improved* A* dengan kombinasi algoritma DDA, apakah algoritma perencanaan dari setiap *node start* ke *node end* aman dari tabrakan terhadap *obstacle* atau tidak. Terdapat dua buah fungsi untuk melakukan perencanaan metode *Improved* A*, antara lain yaitu "**path_normalize**" dan "**dda_algorithm**". Fungsi "**path_normalize**" digunakan untuk menentukan posisi koordinat titik awal tujuan robot dan titik akhir tujuan robot. Sedangkan untuk

fungsi "**dda_algorithm**" berfungsi sebagai kalkulasi perhitungan dari titik tujuan robot awal hingga ke titik tujuan robot akhir dengn 2.5, 2.6, 2.7 dan 2.8 pada algoritma DDA. Output dari fungsi bersangkutan mengembalikan status berupa *node* yang dianggap aman untuk dinormalisasikan atau tidak, ketika *node* bersangkutan aman maka akan melakukan proses penambahan iterasi pada program, sedangkan ketika *node* bersangkutan tidak aman untuk dilalui maka menambahkan data *node* sebelumnya ke dalam variabel output perencanaan algoritma *Improved* A*, dan menjadikan titik *node* sebelumnya sebagai acuan untuk titik koordinat *start* robot yang akan dinormalisasikan. Berikut merupakan representasi dari *source code* program.

```
def path_normalize(self):
     while(self.inc < self.length):
          new_len = len(self.new_path) - 1
          start_point = self.new_path[new_len]
          end_point = self.last_path[self.inc]
          counter = self.dda_algorithm(start_point, end_point)
          if(counter):
              if(self.inc == 0 or self.last_inc == self.inc):
                  self.inc += 1
              new_node = self.last_path[self.inc - 1]
              self.new_path.append(new_node)
              self.inc = len(self.last_path) -
              (len(self.last_path) - self.inc)
              self.last_inc = self.inc
          else:
              self.inc += 1

     else:
          self.new_path.pop(0)
          self.new_path += [self.endP]


    return self.new_path


def dda_algorithm(self, start_pt, end_pt):
     try:
          condition = False
          x1, y1 = start_pt
          x2, y2 = end_pt

          dx = x2-x1
          dy = y2-y1

          step = 0
          if(abs(dx) > abs(dy)):
               step = abs(dx)
          else:
               step = abs(dy)

          step = np.rint(step).astype(int)
          if(step <= 0.):
               step = 1
```

```
          xinc = dx/step
          yinc = dy/step

          for _ in range(step):
               x1 += xinc
               y1 += yinc
               point = (x1, y1)
               obs_total = classKirim.obs
               if(classKirim.isActiveSimulasi == 1):
                    obs_total=classKirim.obs + classKirim.obsO

               for obs in obs_total:
                    new_obs = np.array(obs)
                    new_point = np.array(point)
                    err = new_obs - new_point
                    err = np.linalg.norm(err)
                    if(err <= self.error_obs):
                         condition = True
                         break

          return condition

     except Exception as e:
          print(e)
          self.new_path.append(start_pt)
```

5.1.4 Implementasi Perhitungan Akurasi Metode

A. Pengujian Akurasi Metode

Pengujian akurasi metode digunakan sebagai pengukuran akurasi metode *Impro*ved A* pada percobaan yang dilakukan terhadap rute perencanaan robot. Pengujian ini dilakukan dengan mengecek apakah setiap percobaan yang dilakukan pada perencanaan rute robot menggunakan algoritma *Improved* A* mengalami tabrakan atau tidak terhadap robot lawan atau *obstacle*. Pengujian akurasi yang dilakukan memiliki beberapa opsi dengan memilih model simulasi pada percobaan yang dilakukan, yaitu percobaan secara *hardware* atau percobaan secara simulasi. Berikut merupakan contoh *source code* program pada aplikasi untuk menghitung akurasi metode terhadap tabel yang ada di database sistem yang dibuat.

```
def resultAkurasiMetode(self):
     query = ""
     pos = self.chose_akurasi.currentIndex()
     if(pos == 0):
          query = """SELECT COUNT(*) as total, (SELECT COUNT(*) FROM
          tbl_percobaan WHERE akurasi = 0) as tidak_aman, (SELECT
          COUNT(*) FROM tbl_percobaan WHERE akurasi = 1) as aman
          FROM tbl_percobaan"""
     else:
          v = self.chose_akurasi.currentText()
          query = """SELECT COUNT(*) as total, (SELECT COUNT(*) FROM
```

```
            tbl_percobaan WHERE akurasi = 0 AND mode LIKE '%{0}%') as
            tidak_aman, (SELECT COUNT(*) FROM tbl_percobaan WHERE
            akurasi = 1 AND mode LIKE '%{1}%') as aman FROM
            tbl_percobaan WHERE mode LIKE '%{2}%'""".format(v, v, v)

        result = db.get_manual_query(query)
        result = dict(result[0])
        total = result["total"]
```

B. Pengujian Akurasi Persepsi Manusia

Pengujian akurasi persepsi manusia sebagai pengukuran akurasi terhadap rute perencanaan yang dihasilkan oleh algoritma dengan pandangan penilaian persepsi manusia. Akurasi dari metode yang dilakukan memiliki dua pemodelan, yaitu akurasi yang dilakukan terhadap algoritma A* apakah rute perencanaan yang dihasilkan memiliki jarak yang optimal dan efisien, dan pemodelan yang kedua adalah pengecekan jumlah rute atau *node* yang dihasilkan oleh algoritma *Improved* A* apakah efisien atau tidak. Berikut merupakan contoh *source code* program pada aplikasi untuk menghitung akurasi prespsi manusia terhadap tabel yang ada di database sistem yang dibuat.

- Akurasi prespsi manusia terhadap algoritma A*

```
query = """SELECT COUNT(*) as total, (SELECT COUNT(*) FROM
tbl_metode WHERE nilai == 1 AND id_percobaan = {0} AND mode LIKE
'%{1}%') as efisien, (SELECT COUNT(*) FROM tbl_metode WHERE nilai
== 0 AND id_percobaan = {2} AND mode LIKE '%{3}%') as tidak_efisien
FROM tbl_metode WHERE id_percobaan = {4} AND mode LIKE
'%{5}%'""".format(id, "Astar Search", id, "Astar Search", id, "Astar
Search")

astar_result = db.get_manual_query(query)
astar_result = dict(astar_result[0])
akurasi = (astar_result["efisien"] / astar_result["total"]) * 100

e_mutlak = abs(astar_result["total"] - astar_result["efisien"])
e_relatif = (e_mutlak / astar_result["total"]) * 100
```

- Akurasi prespsi manusia terhadap algoritma *Improved* A*

```
query = """SELECT total_sebenarnya,total_presepsi FROM
tbl_uji_presepsi WHERE id_percobaan = {0}""".format(id)

improved_result = db.get_manual_query(query)
improved_result = dict(improved_result[0])

akurasi = (improved_result["total_presepsi"] /
improved_result["total_sebenarnya"]) * 100

e_mutlak=abs(improved_result["total_sebenarnya"]-
improved_result["total_presepsi"])
e_relatif = (e_mutlak / improved_result["total_sebenarnya"]) * 100
```

5.1.3 Implementasi Tampilan Sistem

Implementasi tampilan *user interface* dari sistem sesuai dengan perancangan desain tampilan yang dilakukan sebelumnya, sebagai berikut:

A. Tampilan Utama Aplikasi

Berikut ini merupakan implementasi dari tampilan utama aplikasi. Pada halaman utama ini terdapat menu bar, *button, combo box, toggle button,* dan *image view* yang akan digunakan dalam proses perencanaan rute robot secara simulasi maupun langsung. Proses perencanaan rute robot terdiri dari tahap penentuan koordinat titik tujuan akhir robot, penentuan titik koordinat robot lawan atau *obstacle* secara statis dengan menggunakan *mouse event* pada *image view* yang telah disediakan. Setelah itu algoritma akan merencanakan rute berdasarkan informasi koordinat yang didapatkan tadi. Setelah perencanaan jalur menggunakan algoritma selesai, langkah selanjutnya yaitu simulasi gerakan robot secara *real time* pada *image view* sesuai dengan *node* atau rute yang telah dinormalisasikan. Proses perencanaan rute pada setiap percobaan akan dimasukkan ke dalam database. Berikut ini merupakan representasi dari gambar tampilan utama aplikasi.

Gambar 5.10 Implementasi Tampilan Utama Aplikasi

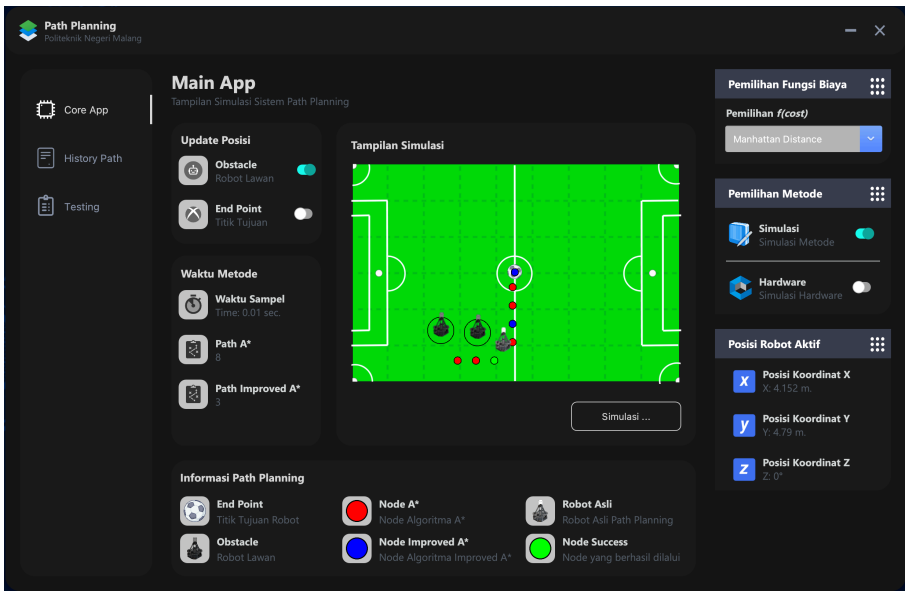
B. Tampilan Perencanaan Rute Algoritma

Perencanaan rute algoritma merupakan langkah awal dari sistem yang akan dijalankan setelah penentuan koordinat titik robot lawan dan koordinat titik tujuan akhir robot. Pada perencanaan rute algoritma terdapat pemilihan fungsi biaya yaitu *Euclidean Distance* dan *Manhattan Distance*, dan juga terdapat pemilihan model dari simulasi robot secara *virtual* atau secara langsung dengan terkoneksi melalui

socket. Hasil perencanaan rute algoritma akan digambarkan sebuah *node* berwarna merah dan biru pada *image view.* Berikut ini merupakan representasi dari gambar tampilan perencanaan rute algoritma.

Gambar 5. 11 Implementasi Tampilan Perancangan Algoritma

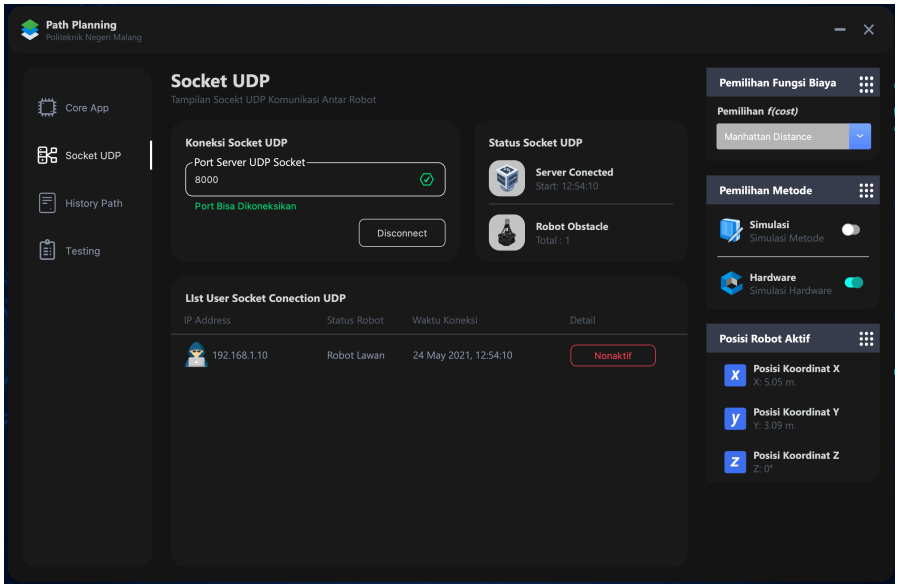
Setelahlah proses perencanaan rute algoritma selesai, maka langkah selanjutnya adalah mensimulasikan pergerakan robot secara *real time* terhadap titik titik koordinat tujuan normalisasi rute atau *node* dengan warna biru yang dihasilkan oleh algoritma *Improved* A*. Rute yang berhasil dilalui oleh robot maka akan direpresentasikan dengan warna hijau. Gambar 5.12 merupakan representasi simulasi robot secara *real time*.



Gambar 5.12 Implementasi Tampilan Pergerakan Robot *Real Time*

C. Tampilan Data *User Socket* UDP

Penerapan perencanaan rute robot yang dilakukan secara langsung atau bukan secara *virtual* diharuskan untuk memulai koneksi terhadap *socket programming User Datagram Protocol* (UDP) dengan model "*broadcast*". Koneksi *socket* digunakan untuk mengambil informasi koordinat robot asli sesuai dengan *port* yang ditentukan pada aplikasi sesuai dengan inputan *user.* Proses pengiriman rute hasil normalisasi dari algoritma *Improved* A* dikirimkan ke robot dengan menggunakan *socket*. Hasil dari pengiriman rute perencanaan ke robot akan disimulasikan pada *image view* dengan melakukan pembacaan posisi secara *real time* robot asli atau langsung. Gambar 5.13 merepresentasikan tampilan data koneksi *socket* UDP yang terkoneksi pada aplikasi. Button detail pada aplikasi menyatakan status pemilihan robot yang akan dilakukan perencanaan rute dengan menggunakan algoritma *Improved* A*. "Nonaktif" menyatakan status robot lawan atau *obstacle,* sedangkan "Aktif" menyatakan status robot yang digunakan untuk menjalankan algoritma *Improved* A*.

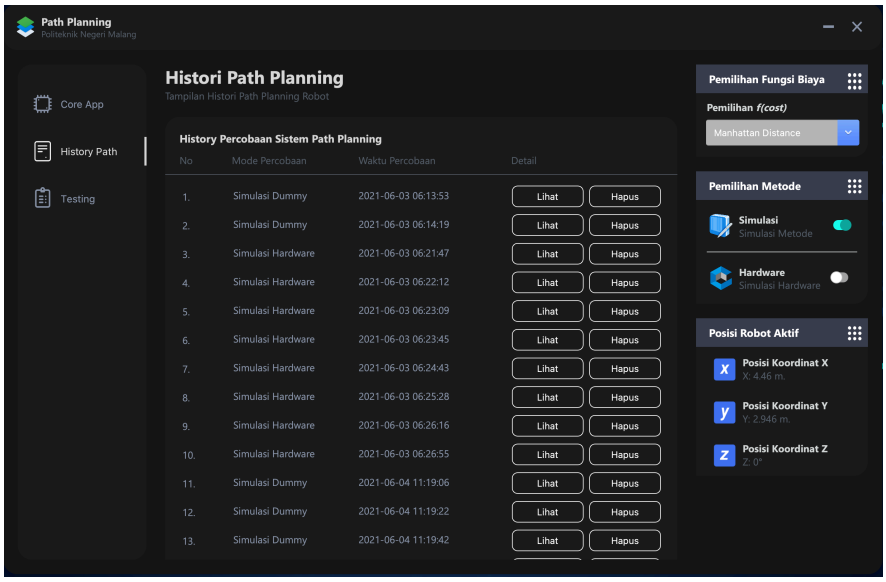


Gambar 5.13 Implementasi Tampilan Data *User Socket* UDP

D. Tampilan Data Hasil Percobaan Perencanaan Rute

Setelah hasil simulasi pergerakan robot dilakukan secara *real time* pada *image view*. Data setiap percobaan yang dilakukan terhadap perencanaan rute menggunakan metode *Improved* A* akan dimasukkan ke dalam database sesuai dengan perancangan database penelitian ini. Hasil dari setiap percobaan yang

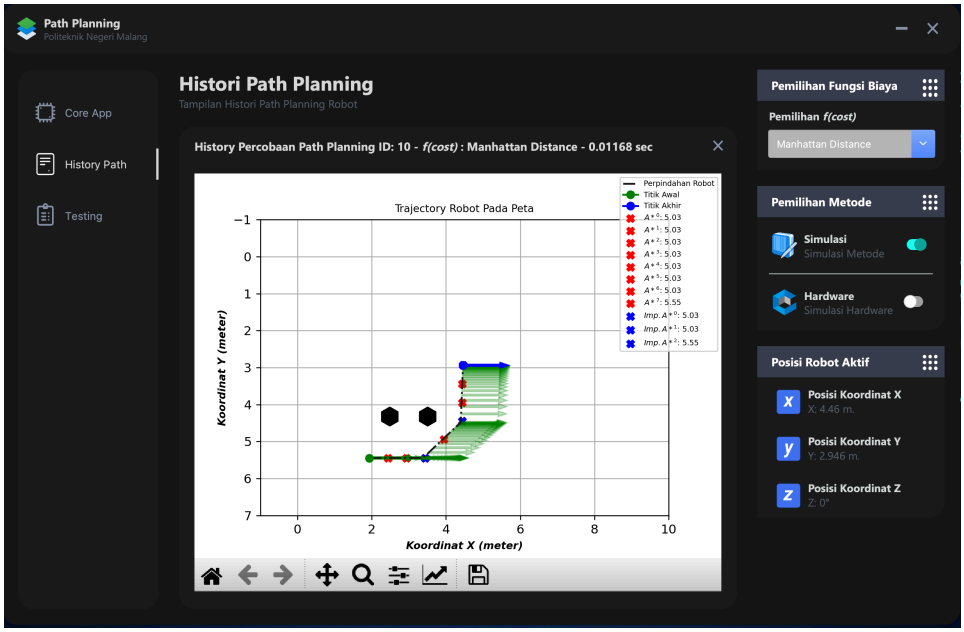
dilakukan akan ditampilkan seperti pada gambar dibawah ini sesuai dengan waktu percobaan yang dilakukan.



Gambar 5.14 Implementasi Tampilan Data Percobaan Perencanaan Rute

E. Tampilan Grafik Simulasi Pergerakan Robot

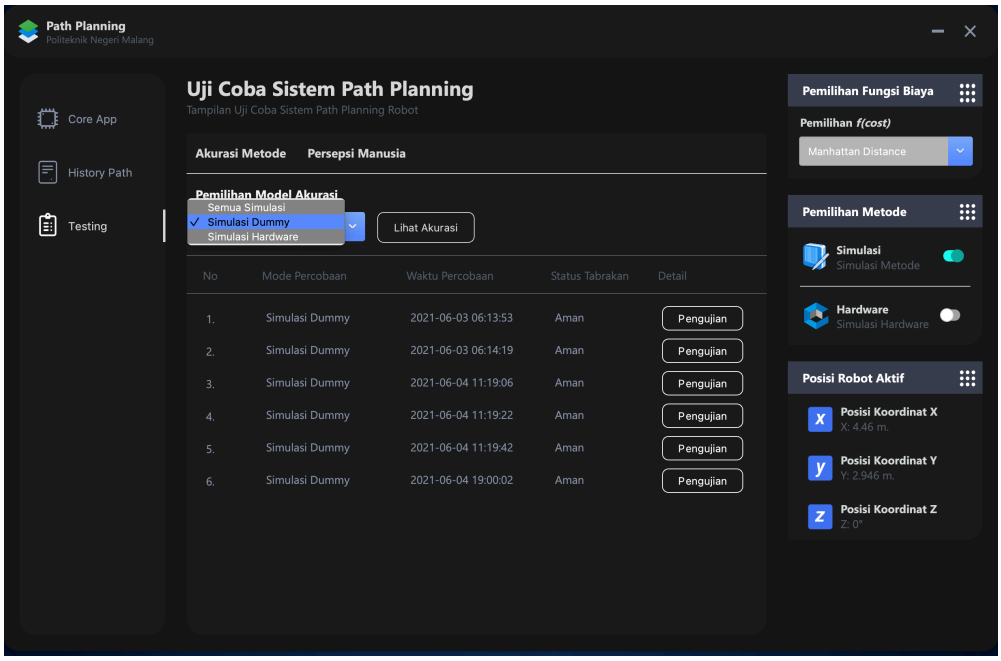
Hasil dari setiap percobaan perencanaan rute robot yang dilakukan oleh algoritma *Improved* A* dipetakan dalam bentuk grafik. Setiap titik koordinat robot, mulai dari posisi koordinat robot awal, posisi koordinat robot lawan, posisi koordinat rute pencarian A*, posisi koordinat normalisasi rute *Improved* A*, posisi koordinat pergerakan robot, dan posisi tujuan akhir dari robot akan ditampilkan seperti gambar dibawah ini.



Gambar 5.15 Implementasi Tampilan Grafik Simulasi Pergerakan Robot

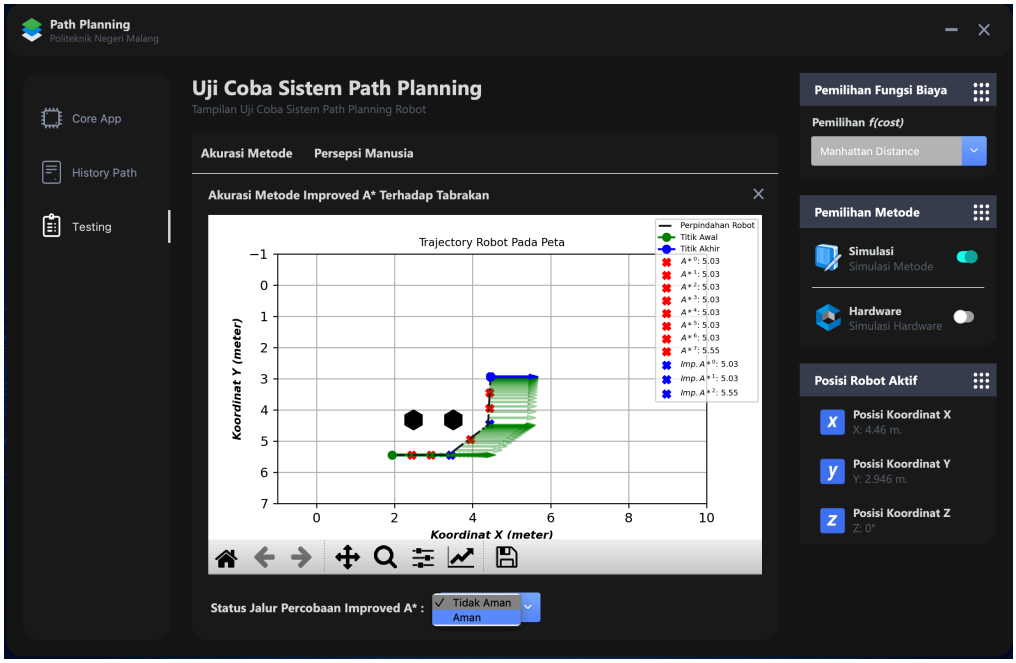
F. Tampilan Data Pengujian Akurasi Metode

Tampilan data pengujian akurasi metode menyatakan pengujian terhadap perencanaan rute yang dilakukan pada setiap percobaan. Output dari pengujian akurasi metode *Improved* A* merepresentasikan apakah jalur yang dilewati oleh robot pada saat simulasi secara *real time* mengalami tabrakan terhadap robot lawan atau *obstacle* atau tidak terjadi tabrakan. Hasil akurasi metode terhadap simulasi yang dilakukan akan direpresentasikan dengan diagram *chart pie* untuk mengetahui akurasi persentase keberhasilan metode terhadap tabrakan dan *error relative* dari percobaan bersangkutan. Gambar 5.16 merepresentasikan output data percobaan terhadap uji coba akurasi metode *Improved* A*.



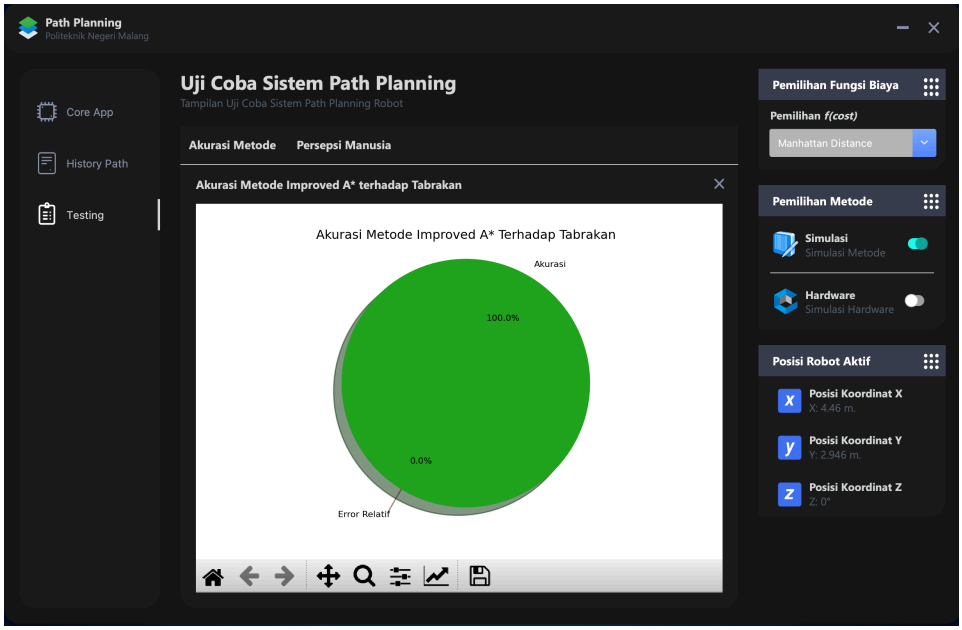
Gambar 5.16 Implementasi Tampilan Data Pengujian Akurasi Metode

Kemudian *user* dihadapkan dalam pengujian akurasi metode terhadap percobaan yang dipilih pada *button* "Pengujian". Gambar dibawah ini merepresentasikan output terhadap status perencanaan rute pada setiap percobaan. Status aman menyatakan bahwa simulasi yang dilakukan tidak mengalami tabrakan, sedangkan status tidak aman menyatakan bahwa simulasi perencanaan rute yang dilakukan mengalami tabrakan.



Gambar 5.17 Implementasi Pengujian Akurasi Metode *Improved* A*

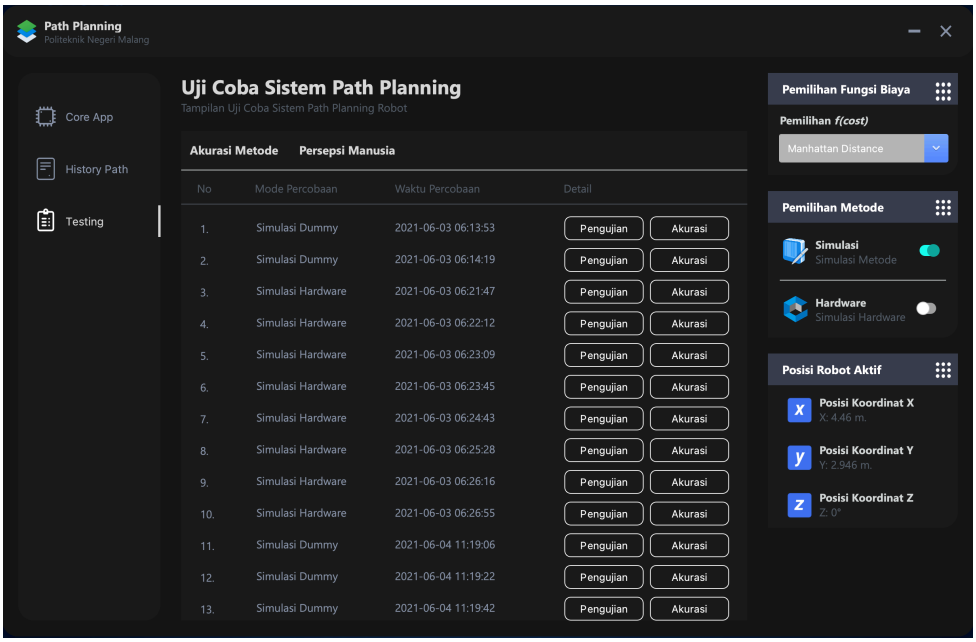
Setelah melakukan proses pengujian akurasi metode, output dari hasil pengujian pada setiap percobaan akan dihitung sesuai dengan status percobaan yang tersimpan dalam database. Gambar 5.18 merepresentasikan tampilan output terhadap perhitungan akurasi persentase keberhasilan metode dan persentase nilai *error relative* perencanaan rute algoritma *Improved* A* dengan pemilihan model simulasi robot secara *virtual.*



Gambar 5.18 Implementasi Tampilan Persentase Akurasi Metode

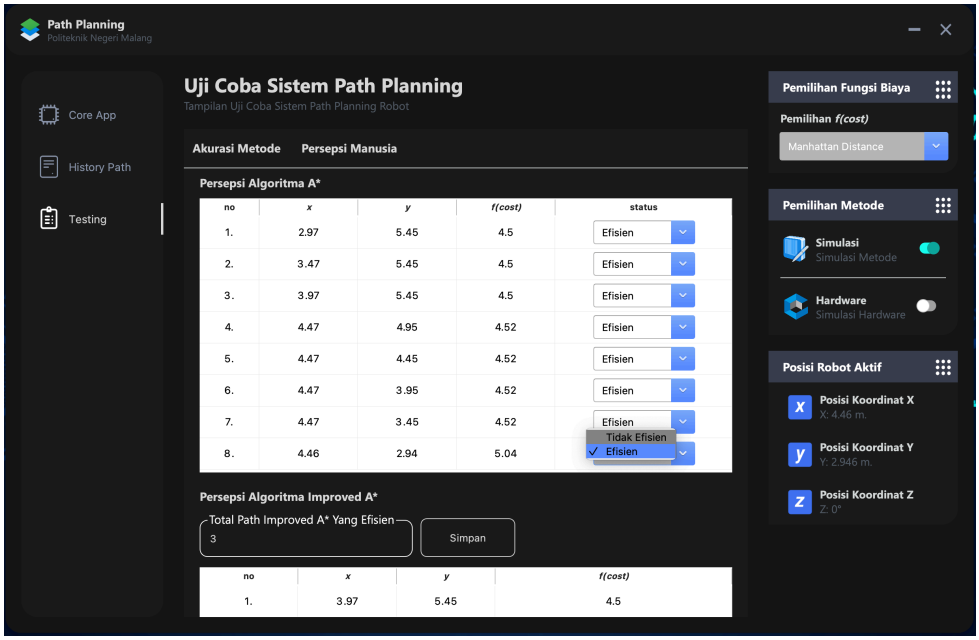
G. Tampilan Data Pengujian Persepsi Manusia

Tampilan data pengujian persepsi manusia menyatakan pengujian efisiensi hasil rute perencanaan terhadap perencanaan rute yang dihasilkan oleh algoritma A* dan *Improved* A*. Output pengujian metode A* menyatakan apakah setiap *node* atau rute yang dihasilkan oleh algoritma menyatakan pemilihan rute yang efisien menurut anotasi atau persepsi manusia, sedangkan output dari pengujian metode *Improved* A* menyatakan jumlah total seharusnya rute yang dinormalisasikan menurut anotasi atau prespsi manusia. Hasil persentase akurasi terhadap simulasi yang dilakukan pada setiap percobaan akan direpresentasikan dengan diagram *chart pie.* Gambar dibawah ini merupakan implementasi dari tampilan data pengujian setiap percobaan yang tersimpan dalam database.



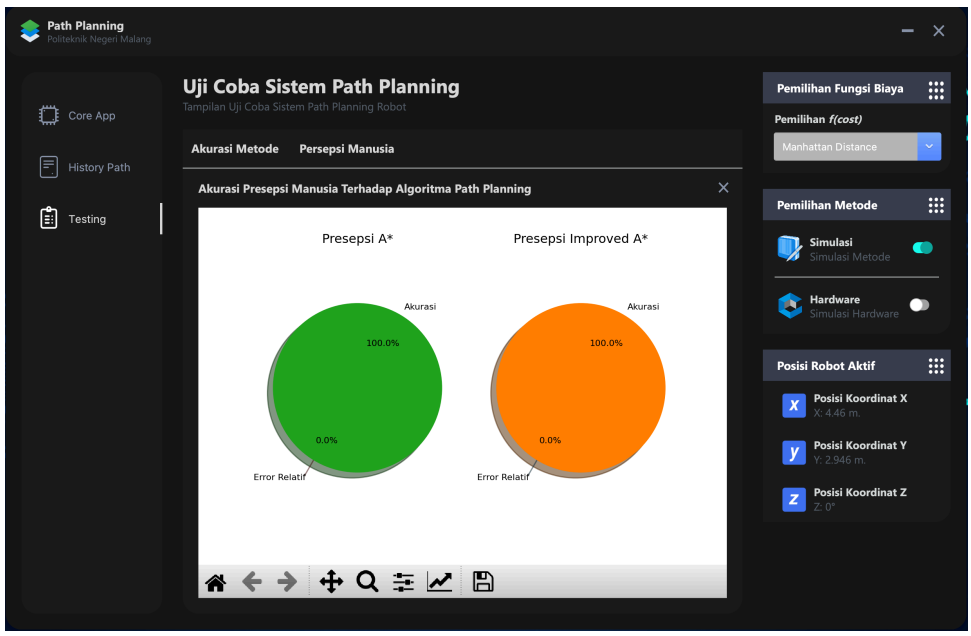
Gambar 5.19 Implementasi Tampilan Data Pengujian Prespsi Manusia

Kemudian *user* dihadapkan dalam pengujian akurasi prespsi manusia terhadap percobaan yang dipilih pada *button* “Pengujian”. Gambar dibawah ini merepresentasikan output terhadap efisiensi pemilihan rute A* dan total normalisasi *Improved* A* menurut prespesi atau anotasi manusia pada setiap percobaan.



Gambar 5.20 Implementasi Pengujian Prespsi Manusia terhadap Algoritma

Setelah melakukan proses pengujian akurasi persepsi manusia terhadap percobaan yang dipilih, output dari hasil pengujian akan dihitung sesuai dengan data yang tersimpan dalam database. Gambar 5.18 merepresentasikan implementasi tampilan terhadap perhitungan akurasi persentase efisiensi metode A* dan *Improved* A* dalam perencanaan rute menurut persepsi manusia.



Gambar 5.21 Implementasi Tampilan Persentase Akurasi Persepsi Manusia

5.2 Pengujian

Pengujian dilakukan untuk mengetahui aplikasi "Sistem *Path Planning* pada Robot Sepak Bola Beroda Politeknik Negeri Malang" sesuai dengan tujuan pembuatan sistem dalam penelitian dan berjalan sesuai fungsinya. Metode pengujian sistem terbagi menjadi dua macam yaitu pengujian sistem dan pengujian akurasi.

5.2.1 Pengujian Sistem

Pengujian sistem bertujuan untuk memastikan bahwa setiap fungsi pada aplikasi sesuai dengan yang diperlukan dan berjalan dengan benar. Pengujian ini dilakukan dengan menggunakan metode *black box.* Pengujian *black box* digunakan untuk menemukan kesalahan aplikasi yang sedang diuji dan untuk mengetahui apakah seluruh fungsi dapat berjalan dengan baik.

a. Pengujian *black box* tampilan utama aplikasi

Pengajuan ini dilakukan pada seluruh fitur *toolbox* yang tersedia pada tampilan utama aplikasi. Berikut ini merupakan tabel pengujian *black box* pada tampilan utama aplikasi.

Tabel 5.1 Pengujian *black box* Tampilan Utama Aplikasi

No.	Skenario Pengujian	Hasil yang diharapkan	Hasil Pengujian	Kesimpulan
1.	Mengaktifkan *Toggle button obstacle*	Dapat membuat *obstacle virtual* pada *image view* tampilan simulasi secara statis dengan menggunakan *mouse event click.*	Sesuai	Berhasil
2.	Menonaktifkan *Toggle button obstacle*	Tidak dapat membuat *obstacle virtual* pada *image view* tampilan simulasi.	Sesuai	Berhasil
3.	Mengaktifkan *Toggle button end point*	Dapat merubah posisi dari titik tujuan akhir robot pada *image*	Sesuai	Berhasil

		view tampilan simulasi.		
4.	Menonaktifkan *Toggle button endpoint*	Tidak dapat merubah posisi dari titik tujuan akhir robot pada *image view* tampilan simulasi.	Sesuai	Berhasil
5.	Klik *button Start Simulasi*	Melakukan simulasi perhitungan metode dan menampilkan hasil rute pada *image view* tampilan simulasi dengan sebuah *node.*	Sesuai	Berhasil
6.	Klik *button Kirim Ke Robot > Yes*	Melakukan pengirim data hasil perencanaan rute algoritma untuk melakukan simulasi robot secara *real time*.	Sesuai	Berhasil
7.	Pilih *Combobox* pemilihan fungsi biaya *heuristic*	Melakukan perubahan fungsi biaya perencanaan rute algoritma.	Sesuai	Berhasil
8.	Mengaktifkan *Toggle button simulasi*	Melakukan perencanaan rute robot terhadap robot *virtual.*	Sesuai	Berhasil
9.	Mengaktifkan *Toggle button hardware*	Melakukan perencanaan rute terhadap robot secara langsung, dan	Sesuai	Berhasil

		mengaktifkan menu komunikasi *socket User Datagram Protocol.*		
10.	*Label view* posisi koordinat robot aktif	Menampilkan posisi koordinat dengan format *x,y,z* secara *real time* pada saat simulasi pergerakan robot terhadap rute perencanaan algoritma yang didapatkan.	Sesuai	Berhasil

b. Pengujian *black box* Menu *Socket* UDP

Pengajuan ini dilakukan pada seluruh fitur *toolbox* yang tersedia pada menu *socket* UDP aplikasi. Berikut ini merupakan tabel pengujian *black box* pada menu *socket* UDP aplikasi.

Tabel 5.2 Pengujian *black box* menu *socket* UDP

No.	**Skenario Pengujian**	**Hasil yang diharapkan**	**Hasil Pengujian**	**Kesimpulan**
1.	*Input* port *socket* UDP	Menampilkan *alert* terhadap nilai input *port socket* oleh user ketika berhasil atau gagal.	Sesuai	Berhasil
2.	Klik *button* Connect	Menghubungkan koneksi *socket* UDP sesuai port yang ditentukan dengan model koneksi *socket* berbasis “broadcast”.	Sesuai	Berhasil

3.	Klik *button* Disconnect	Memutuskan koneksi *socket* UDP.	Sesuai	Berhasil
4.	*View* koneksi *socket* UDP	Menampilkan status koneksi *socket* dari *user* terhadap *socket* aplikasi.	Sesuai	Berhasil
5.	Klik *button* Aktif	Mengganti status robot yang digunakan sebagai penerapan perencanaan rute algoritma.	Sesuai	Berhasil
6.	Klik *button* Nonaktif	Mengganti status robot menjadi robot lawan atau *obstacle.*	Sesuai	Berhasil

c. Pengujian *black box* Menu *History*

Pengajuan ini dilakukan pada seluruh fitur *toolbox* yang tersedia pada menu *history* aplikasi. Berikut ini merupakan tabel pengujian *black box* pada menu *history* pada aplikasi.

Tabel 5.3 Pengujian *black box* menu *history*

No.	**Skenario Pengujian**	**Hasil yang diharapkan**	**Hasil Pengujian**	**Kesimpulan**
1.	Klik *button* Hapus > Yes	Menghapus percobaan yang dilakukan sesuai dengan id percobaan.	Sesuai	Berhasil
2.	Klik *button* Lihat	Menampilkan grafik simulasi perencanaan rute robot sesuai dengan id percobaan.	Sesuai	Berhasil

d. Pengujian *black box* Menu *Testing*

Pengajuan ini dilakukan pada seluruh fitur *toolbox* yang tersedia pada menu *testing* aplikasi. Berikut ini merupakan tabel pengujian *black box* pada menu *testing* pada aplikasi.

Tabel 5.4 Pengujian *black box* menu *testing*

No.	Skenario Pengujian	Hasil yang diharapkan	Hasil Pengujian	Kesimpulan
1.	Klik *button* Akurasi Metode	Menampilkan hasil pengujian terhadap percobaan metode *Improved* A*.	Sesuai	Berhasil
2.	Klik *button* Presepsi Manusia	Menampilkan hasil pengujian terhadap percobaan metode A* dan *Improved* A*.	Sesuai	Berhasil
3.	Pemilihan *Combobox* pada menu Akurasi Metode	Menampilkan hasil pengujian sesuai dengan model simulasi yang dipilih *user.*	Sesuai	Berhasil
4.	Klik *button* Lihat Akurasi pada menu Akurasi Metode	Menampilkan akurasi metode *Improved* A* terhadap pemilihan model simulasi dalam bentuk *diagram pie chart*.	Sesuai	Berhasil
5.	Klik *button* Pengujian pada menu Akurasi Metode	Menampilkan *form* status pengujian akurasi metode *Improved* A*	Sesuai	Berhasil
6.	Pemilihan *Combobox* status jalur percobaan pada	Melakukan pembaharuan data pengujian akurasi akurasi metode	Sesuai	Berhasil

	menu Akurasi Metode	*Improved* A* pada setiap percobaan yang dipilih.		
7.	Klik *button* Akurasi pada menu Presepsi Manusia	Menampilkan akurasi persepsi manusia terhadap metode *Improved* A* dan A* dalam bentuk *diagram pie chart*.	Sesuai	Berhasil
8.	Klik *button* Pengujian pada menu Presepsi Manusia	Menampilkan *form* status pengujian akurasi metode A* dan *Improved* A* terhadap persepsi manusia.	Sesuai	Berhasil
9.	Pemilihan *Combobox* status efisiensi *node* pada menu Presepsi Manusia	Melakukan pembaharuan data pengujian akurasi persepsi manusia terhadap perencanaan rute metode A* pada setiap percobaan yang dipilih.	Sesuai	Berhasil
10.	*Form input* total efisiensi rute perencanaan algoritma *Improved* A* > Klik *button* Simpan	Melakukan pembaharuan data pengujian akurasi persepsi manusia terhadap total hasil perencanaan metode *Improved* A*.	Sesuai	Berhasil

5.2.2 Pengujian Akurasi

Pengujian akurasi bertujuan untuk mengetahui tingkat keberhasilan dari sistem dalam merencanakan rute perjalanan yang akan dilewati oleh robot. Dalam pengujian akurasi dilakukan dengan beberapa sampel percobaan dengan nilai jarak *trehsold* tabrakan yang berbeda dengan rincian sebagai berikut:

A. Pengujian Akurasi Metode *Improved* A*

Pengujian akurasi metode *Improved* A* merepresentasikan persentase akurasi dari perencanaan yang dihasilkan oleh algoritma apakah rute yang dipilih aman dari tabrakan terhadap robot lawan atau *obstacle*.

Tabel 5.5 Pengujian akurasi terhadap simulasi *virtual* dengan jarak *threshold* 0.6375 meter

No.	***start point***			***end point***			**Total *obstacle***	**Waktu Pencarian (*seconds)***	**status rute perencanaan**
	x	***y***	***z***	***x***	***y***	***z***			
1.	4.47	2.89	0º	0.5	0.46	0º	-	0.02441	Aman
2.	2.43	5.50	0º	4.5	2.96	0º	1	0.01655	Aman
3.	2.43	5.50	0º	4.5	2.96	0º	2	0.01089	Aman
4.	2.43	5.50	0º	4.5	2.96	0º	3	0.02132	Aman
5.	4.48	2.90	0º	2.46	5.47	0º	3	0.0305	Aman

Tabel 5.6 Pengujian akurasi terhadap simulasi *virtual* dengan jarak *threshold* 0.810 meter

No.	***start point***			***end point***			**Total *obstacle***	**Waktu Pencarian (*seconds)***	**status rute perencanaan**
	x	***y***	***z***	***x***	***y***	***z***			
1.	4.47	2.89	0º	0.5	0.46	0º	-	0.02573	Aman
2.	2.43	5.50	0º	4.5	2.96	0º	1	0.0155	Aman
3.	2.43	5.50	0º	4.5	2.96	0º	2	0.0063	Aman
4.	2.43	5.50	0º	4.5	2.96	0º	3	0.01596	Aman
5.	4.48	2.90	0º	2.46	5.47	0º	3	0.05177	Aman

Tabel 5.7 Pengujian nilai *error* simulasi *virtual* akurasi metode

No.	***Threshold***	**Jumlah Sampel**	**Jumlah Sampel Berhasil**	**Jumlah Sampel Berhasil**	***Error***
1.	0.6375	5	5	0	0%
2.	0.810	5	5	0	0%

Pada rekapitulasi pengujian di atas menunjukkan jumlah total keberhasilan perencanaan rute dengan nilai *threshold* berbeda dalam 5 sampel percobaan dengan model penerapan terhadap robot *virtual* atau secara simulasi. Untuk mengetahui tingkat akurasi dari sistem dapat dihitung dengan menggunakan rumus yang direpresentasikan pada persamaan 3.5. Tabel dibawah ini merepresentasikan output dari akurasi percobaan perencanaan rute algoritma *Improved* A*.

Tabel 5.8 Akurasi Metode *Improved* A* simulasi *virtual*

No.	***Threshold***	**Jumlah Sampel**	***Error***	**Akurasi**
1.	0.6375	5	0%	100%
2.	0.810	5	0%	100%

Tabel 5.9 Pengujian akurasi terhadap simulasi *hardware* dengan jarak *threshold* 0.6375 meter

No.	***start point***			***end point***			**Total *obstacle***	**Waktu Pencarian (*seconds)***	**status rute perencanaan**
	x	***y***	***z***	***x***	***y***	***z***			
1.	2.53	5.76	0º	4.46	2.99	0º	2	0.01769	Tidak Aman
2.	1.0	5.35	0º	4.5	2.95	0º	3	0.04197	Aman
3.	0.65	0.41	0º	4.5	5.5	0º	3	0.0457	Aman
4.	4.37	5.75	0º	0.73	0.81	0º	3	0.05834	Aman
5.	0.4	0.98	0º	4.2	5.54	0º	3	0.5549	Tidak Aman
6.	4.9	4.67	0º	5.23	3.0	0º	-	0.00251	Aman
7.	5.25	3.0	0º	0.525	0.625	0º	1	0.02681	Aman
8.	2.34	2.74	0º	2.56	4.56	0º	2	0.02145	Aman

Tabel 5.10 Pengujian akurasi terhadap simulasi *hardware* dengan jarak *threshold* 0.810 meter

No.	***start point***			***end point***			**Total *obstacle***	**Waktu Pencarian (*seconds)***	**status rute perencanaan**
	x	***y***	***z***	***x***	***y***	***z***			
1.	2.45	4.90	0º	3.8	2.98	0º	-	0.0052	Aman
2.	3.8	2.98	0º	2.45	4.90	0º	-	0.02246	Aman
3.	2.45	4.92	0º	4.45	2.94	0º	1	0.3762	Aman

4.	4.45	2.94	0º	2.45	4.92	0º	2	0.12945	Tidak Aman
5.	2.45	4.92	0º	4.45	2.94	0º	2	0.01371	Aman
6.	4.22	3.05	0º	2.60	5.55	0º	3	0.03859	Aman
7.	2.58	5.65	0º	4.27	1.30	0º	3	0.28735	Aman
8.	4.22	1.46	0º	1.55	4.15	0º	3	0.0528	Aman

Tabel 5.11 Pengujian nilai *error* simulasi *hardware* akurasi metode

No.	***Threshold***	**Jumlah Sampel**	**Jumlah Sampel Berhasil**	**Jumlah Sampel Berhasil**	***Error***
1.	0.6375	8	6	2	25%
2.	0.810	8	7	1	12.5%

Pada rekapitulasi pengujian di atas menunjukkan jumlah total keberhasilan perencanaan rute dengan nilai *threshold* berbeda dalam 8 sampel percobaan dengan model penerapan terhadap robot *virtual* atau secara simulasi. Untuk mengetahui tingkat akurasi dari sistem dapat dihitung dengan menggunakan rumus yang direpresentasikan pada persamaan 3.5. Tabel dibawah ini merepresentasikan output dari akurasi percobaan perencanaan rute algoritma *Improved* A*.

Tabel 5.12 Akurasi Metode Improved A* simulasi *hardware*

No.	***Threshold***	**Jumlah Sampel**	***Error***	**Akurasi**
1.	0.6375	8	25%	75%
2.	0.810	8	12.5%	87.5%

B. Pengujian Persepsi Manusia A*

Pada rekapitulasi pengujian menunjukkan jumlah total efisiensi perencanaan rute menggunakan metode A* dengan nilai *threshold* berbeda dalam 3 sampel percobaan dengan model penerapan terhadap robot *virtual* atau secara simulasi. Untuk mengetahui tingkat akurasi dari sistem dapat dihitung dengan menggunakan rumus yang direpresentasikan pada persamaan 3.6. Tabel dibawah ini merepresentasikan output dari akurasi efisiensi percobaan perencanaan rute algoritma A* terhadap persepsi manusia atau anotasi manusia.

Tabel 5.13 Pengujian persepsi manusia metode A* terhadap simulasi *virtual* dengan jarak *threshold* 0.6375 meter

No.	*start point*			*end point*			Total rute algoritma	Total rute Persepsi Manusia	Akurasi
	x	***y***	***z***	***x***	***y***	***z***			
1.	4.47	2.89	0º	0.5	0.46	0º	9	9	100 %
2.	2.43	5.50	0º	4.5	2.96	0º	6	6	100 %
3.	2.43	5.50	0º	4.5	2.96	0º	7	7	100 %

Tabel 5.14 Pengujian persepsi manusia metode A* terhadap simulasi *virtual* dengan jarak *threshold* 0.810 meter

No.	*start point*			*end point*			Total rute algoritma	Total rute Persepsi Manusia	Akurasi
	x	***y***	***z***	***x***	***y***	***z***			
1.	4.47	2.89	0º	0.5	0.46	0º	9	9	100%
2.	2.43	5.50	0º	4.5	2.96	0º	7	7	100%
3.	2.43	5.50	0º	4.5	2.96	0º	7	7	100%

C. Pengujian Persepsi Manusia *Improved* A*

Pada rekapitulasi pengujian menunjukkan jumlah total efisiensi perencanaan rute menggunakan metode *Improved* A* dengan nilai *threshold* berbeda dalam 3 sampel percobaan dengan model penerapan terhadap robot *virtual* atau secara simulasi. Untuk mengetahui tingkat akurasi dari sistem dapat dihitung dengan menggunakan rumus yang direpresentasikan pada persamaan 3.6. Tabel dibawah ini merepresentasikan output dari akurasi efisiensi percobaan perencanaan rute algoritma *Improved* A* terhadap persepsi manusia atau anotasi manusia.

Tabel 5.15 Pengujian persepsi manusia metode *Improved* A* terhadap simulasi *virtual* dengan jarak *threshold* 0.6375 meter

No.	*start point*			*end point*			Total rute algoritma	Total rute Persepsi Manusia	Akurasi
	x	***y***	***z***	***x***	***y***	***z***			
1.	4.47	2.89	0º	0.5	0.46	0º	1	1	100 %
2.	2.43	5.50	0º	4.5	2.96	0º	2	2	100 %
3.	2.43	5.50	0º	4.5	2.96	0º	2	2	100 %

Tabel 5.16 Pengujian persepsi manusia metode *Improved* A* terhadap simulasi *virtual* dengan jarak *threshold* 0.810 meter

No.	***start point***			***end point***			**Total rute algoritma**	**Total rute Persepsi Manusia**	**Akurasi**
	x	***y***	***z***	***x***	***y***	***z***			
1.	4.47	2.89	0º	0.5	0.46	0º	1	1	100%
2.	2.43	5.50	0º	4.5	2.96	0º	3	2	66.7%
3.	2.43	5.50	0º	4.5	2.96	0º	3	2	66.7%