# BAB III

# METODE PENELITIAN

Untuk pengumpulan data yang diperlukan dalam melaksanakan tugas akhir, ada beberapa cara yang telah dilakukan, antara lain:

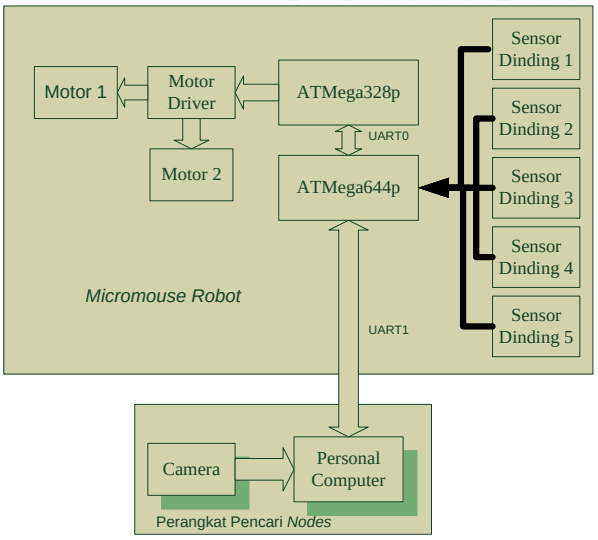
1. Studi kepustakaan

Studi kepustakaan dengan memahami buku referensi, data yang diperlukan untuk menunjang pembuatan alat, baik pada perangkat keras maupun perangkat lunak.

2. Penelitian laboratorium

Dilakukan dengan mengadakan percobaan, pengujian modul-modul serta mengintegrasikan modul tersebut dengan perangkat lunak untuk mengendalikan sistem agar menjadi satu kesatuan yang utuh dan diperoleh hasil yang seoptimal mungkin.

Keseluruhan sistem pada penelitian ini sesuai dengan blok diagram pada gambar 3.1.



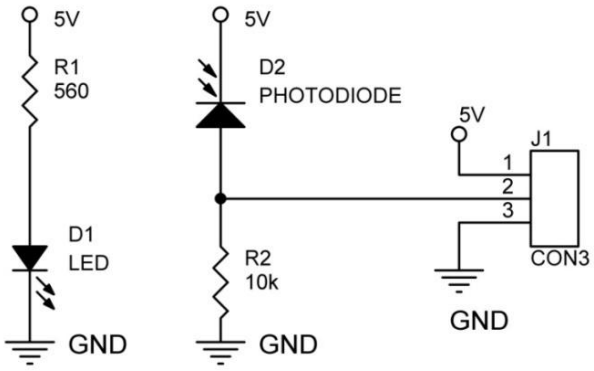

Gambar 3.1 Blok diagram keseluruhan sistem.

Pada perangkat pencari *nodes* citra labirin terdapat *webcam* yang terhubung dengan PC untuk pengambilan citra, citra yang didapat diolah di PC dan mengirim hasilnya ke *Micromouse Robot* menggunakan protokol UART. Pada *Micromouse Robot* terdapat 5 buah sensor dinding yang keseluruhannya terhubung dengan ATMega644P sebagai master dan pengolah data sensor. Hasil pengolahan oleh ATMega644P berupa *error* posisi robot dikirim ke ATMega328P yang juga terhubung dengan menggunakan protokol UART. Pada ATMega328P mempunyai tugas mengatur putaran motor pada roda kiri dan kanan sesuai dengan nilai *error* yang diterima. Driver motor yang terhubung dengan ATMega328P berfungsi sebagai penguat arus untuk menggerakkan motor roda kiri dan kanan.

## 3.1 Perancangan Perangkat Keras

### 3.1.1 Perancangan Sensor Reflektansi IR

Untuk mengukur jarak antara dinding labirin dengan robot maka digunakan sensor reflektansi IR dengan menggunakan IR LED sebagai *transmitter* dan fotodioda sebagai *receiver* yang rangkaiannya dapat ditunjukkan seperti pada gambar 3.2.



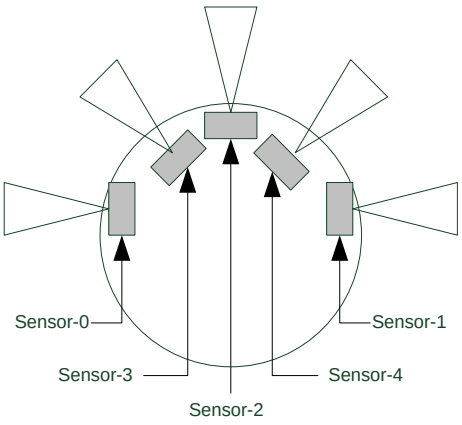

Gambar 3.2 Rangkaian sensor reflektansi.

Pada IR – LED terhubung dengan resistor dengan nilai sebesar 560 Ω sehingga arus yang melalui IR – LED dapat diketahui seperti perhitungan berikut:

$$I(IR) = \frac{5\ V}{560\Omega} = 8{,}93\ mA$$

Prinsip kerja dari penerapan rangkaian seperti pada gambar 3.2 yaitu saat sensor mencapai jarak terdekat dengan dinding maka intensitas cahaya pantulan akan semakin besar sehingga nilai *output* rangkaian tersebut akan semakin besar, begitu juga sebaliknya. Dari rangkaian sensor tersebut nilai *output* yang diharapkan mendekati 0 V saat mencapai jarak terjauh antara sensor dengan dinding dan mendekati 5 V saat mencapai jarak terdekat antara sensor dengan dinding.

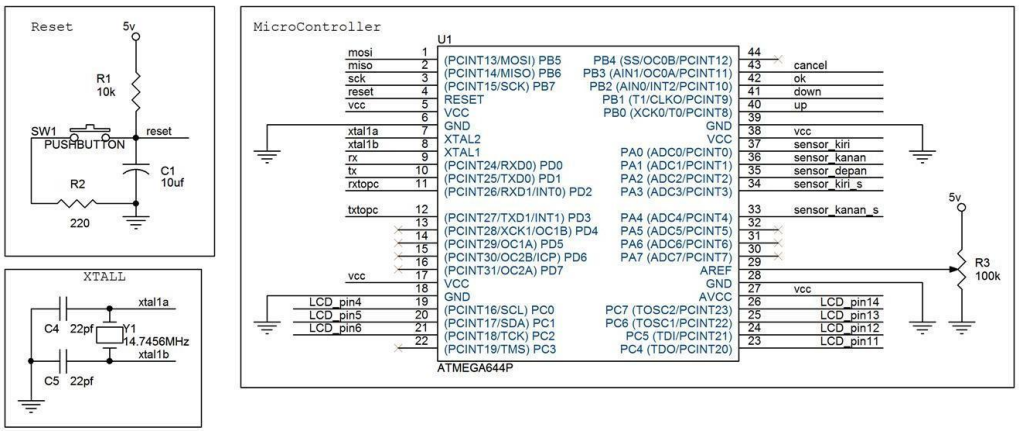
Sensor reflektansi IR yang digunakan sebanyak 5 buah, yang tersusun seperti pada gambar 3.3.




Gambar 3.3 Susunan sensor pada *Micromouse Robot*.

### 3.1.2 Perancangan *Minimum System* Pengendali *Micromouse Robot*

Pada dasarnya 3π didesain sebagai penjejak garis, untuk mengatasi masalah pada lintasan labirin yang berupa dinding diperlukan sensor tambahan yaitu sensor dinding. Antara sensor dinding dengan mikrokontroler pada 3π tidak dapat dihubungkan karena keterbatasan I/O sehingga memerlukan *minimum system* tambahan untuk menangani, mengolah sensor dinding dan mengirim hasilnya ke mikrokontroler pada 3π. Disamping itu *minimum system* tambahan berfungsi untuk menerima data hasil pengolahan citra dari PC. Rangkaian *minimum system* tambahan dapat ditunjukkan seperti pada gambar 3.4.

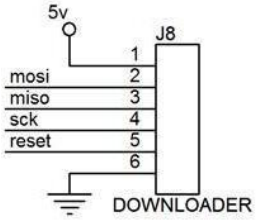



Gambar 3.4 Rangkaian *minimum system* tambahan.

Pada pin VCC terhubung dengan tegangan antara 4.5 – 5.5 V dan pin GND terhubung dengan tegangan 0 V, kedua tegangan tersebut merupakan tegangan teregulasi dari 3π. Pin XTAL1 dan XTAL2 dihubungkan dengan komponen kristal dengan nilai sebesar 11,0592MHz, pemilihan berdasarkan *datasheet* ATMega644p dengan nilai maksimal clock sebesar 20MHz.

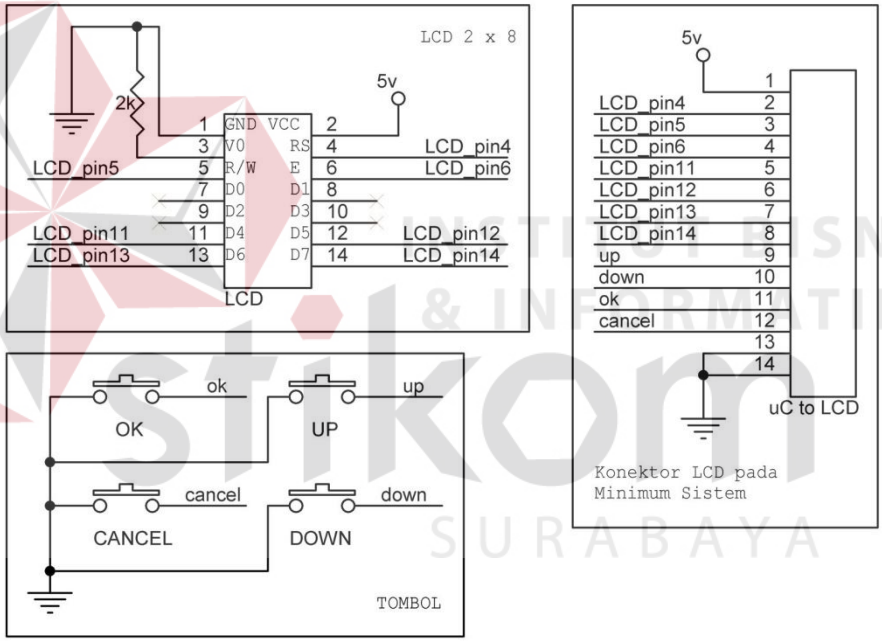
Pin RESET berfungsi untuk masukan reset program secara otomatis atau manual. Pada ATMega644p *reset* bekerja jika terhubung dengan 0V (aktif *low*). Pada saat pertama kali *minimum system* dihidupkan maka rangkaian *reset* akan bekerja secara otomatis, setelah itu rangkaian *reset* harus diaktifkan secara manual. Cara kerja otomatis rangkaian tersebut yaitu saat pertama kali dihidupkan arus yang mengalir mengisi kapasitor sehingga arus pada pin RESET menjadi berkurang jika arus kurang maka pada pin RESET menjadi ~0V (*low*) dan berlahan-lahan akan menjadi high. Begitupun pada cara kerja manual, saat tombol reset ditekan maka arus akan mengalir ke ground sehingga pin RESET menjadi 0V(low), saat tombol reset dilepas maka kapasitor yang yang telah melepas arus akan terisi kembali dan secara berlahan-lahan pin RESET menjadi *high*.

Untuk melakukan proses *downloading* dari komputer ke dalam memori internal mikrokontroler, digunakan kabel *downloader* AVR910 dengan *interface* RS232. Pada ATMega644p *downloader* terhubung dengan pin MOSI, MISO, SCK, RESET yang dapat ditunjukkan seperti pada gambar 3.5. Sedangkan software yang digunakan yaitu Code Vision AVR 2.03.4.



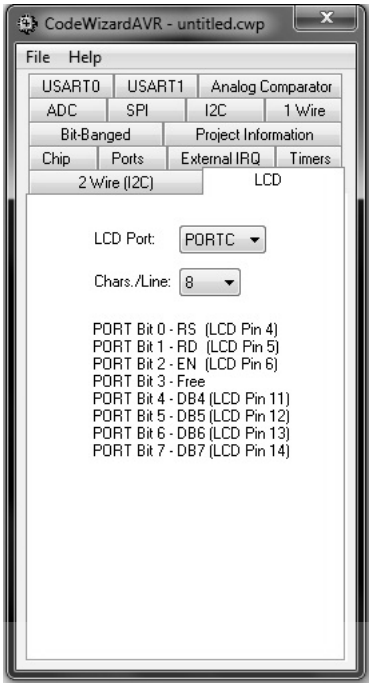

Gambar 3.5 Pin pada mikrokontroler ATMega644P yang terhubung dengan *downloader*.

Agar mempermudah pembacaan data dan melakukan pengaturan pada *Micromouse Robot* maka digunakan display berupa LCD 2×8 dan 4 buah tombol dengan rangkaian seperti pada gambar 3.6.




Gambar 3.6 *Interfacing* LCD dan tombol ke mikrokontroler.

Setelah mempunyai LCD dan sudah dirangkai seperti Gambar 3.3, maka penggunaan LCD dapat diatur pada *wizard* Code Vision AVR 2.03.4 seperti pada Gambar 3.7.




Gambar 3.7 *Wizard window* untuk mengatur LCD.

Setelah melakukan pengaturan LCD pada *wizard* Code Vision AVR 2.03.4, maka LCD dapat diakses dengan penulisan program seperti berikut:

```
lcd_init(8);                    //inisialisasi LCD 8x2
lcd_gotoxy(0,0);                //menempatkan kursor pada kolom 0,
                                //baris 0
lcd_putsf("HALLO…");            //menampilkan string HALLO…
lcd_gotoxy(0,1);                //menempatkan kursor pada kolom 0,
                                //baris 1
```

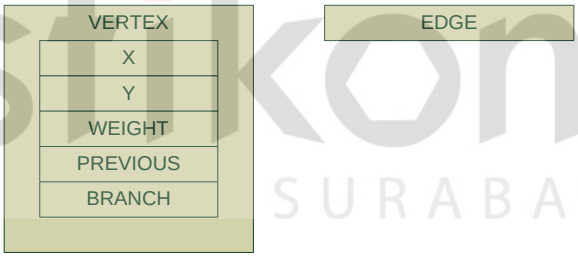
Keempat tombol terhubung dengan PB0, PB1, PB2, dan PB3 pada ATMega644P yang difungsikan sebagai *input* digital dengan *pull-up* internal. Untuk mengatur fungsi PB0 hingga PB3 sebagai *input* digital dengan *pull-up* internal dapat ditunjukkan pada penulisan program seperti berikut:

```
PORTB |= 0x0F;
DDRB &= 0xF0;
```

## 3.2 Perancangan Perangkat Lunak

### 3.2.1 Struktur Data

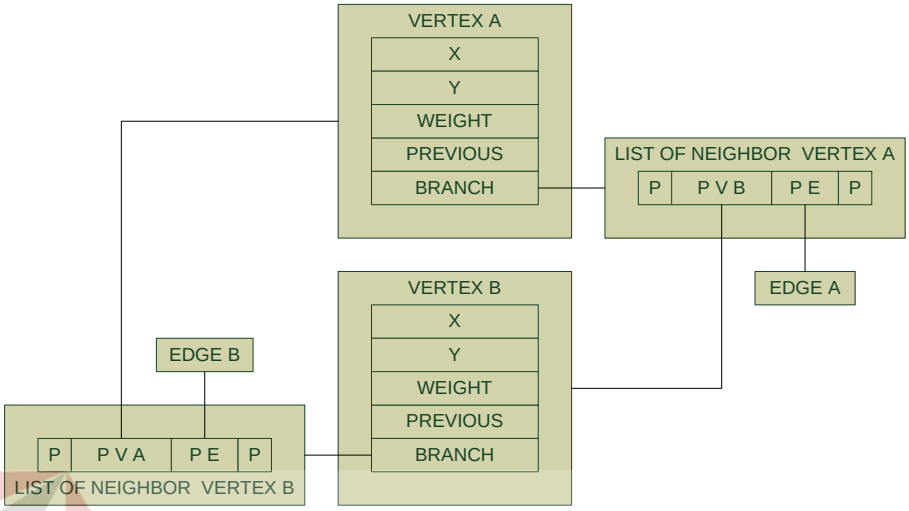
Dalam pemrograman untuk menyelesaikan permasalahan pencarian rute dengan jarak terpendek pada lintasan labirin dapat menggunakan struktur data yang bersifat statis maupun dinamis. Penggunaan struktur data yang bersifat dinamis mempunyai kelebihan dapat menyesuaikan penggunaan tempat pada media penyimpanan sesuai dengan kebutuhan dibandingkan dengan struktur data yang bersifat statis. Dalam algoritma Dijkstra struktur data yang digunakan dalam pengolahan berbentuk graf, dimana sebuah titik (*vertex*) diwakili oleh sebuah *set of data* dan sebuah jarak dari sebuah *vertex* satu ke sebuah *vertex* yang lain (*edge*) diwakili sebuah data yang ilustrasi keduanya dapat ditunjukkan seperti pada gambar 3.8.




Gambar 3.8 Ilustrasi *vertex* dan *edge*.

Pada *vertex* terdiri *x*, *y*, *weight*, *preveious*, dan *branch*. *x* dan *y* digunakan untuk menyimpan koordinat dari titik *vertex* pada citra dari area labirin. *Weight* digunakan untuk menyimpan jarak dari titik *start* menuju ke *vertex* tersebut yang pada awalnya memiliki nilai tak terhingga. *Previous* merupakan sebuah *pointer* yang menunjuk kepada *vertex* sebelumnya. *Branch* merupakan sebuah *pointer* yang menunjuk kepada sebuah *set of data* yang berisi *pointer vertex-vertex* yang

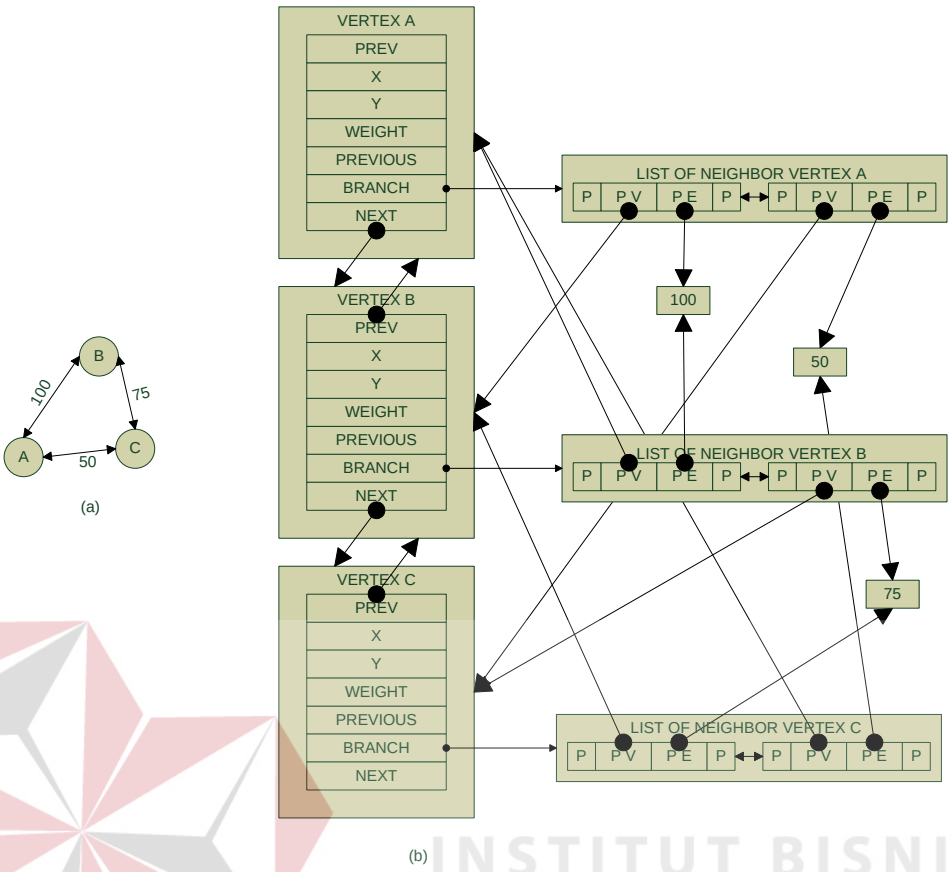
terhubung kepadanya yang dapat diilustrasikan seperti yang ditunjukkan pada gambar 3.9.




Gambar 3.9 Ilustrasi hubungan antar *vertex*.

Pada *vertex* A memiliki daftar *vertex* yang terhubung terhadapnya di dalam *list of neighbor vertex* A. PVB merupakan sebuah *pointer* yang menunjuk pada *vertex B* dan PE merupakan sebuah *pointer* yang menunjuk pada *edge* B. Begitu juga pada *vertex* B memiliki daftar *vertex* yang terhubung terhadapnya di dalam *list of neighbor vertex* B. PVA merupakan sebuah *pointer* yang menunjuk pada *vertex B* dan PE merupakan sebuah *pointer* yang menunjuk pada *edge* A.

Dalam sebuah graf untuk mewakili area labirin maka jumlah *vertex* yang disimpan akan tergantung dari jumlah persimpangan dan jalan buntu yang ada. Secara keseluruhan sebuah graf dengan menggunakan struktur data *Double Linked List* (DLL) dapat diilustrasikan seperti yang ditunjukkan pada gambar 3.10.

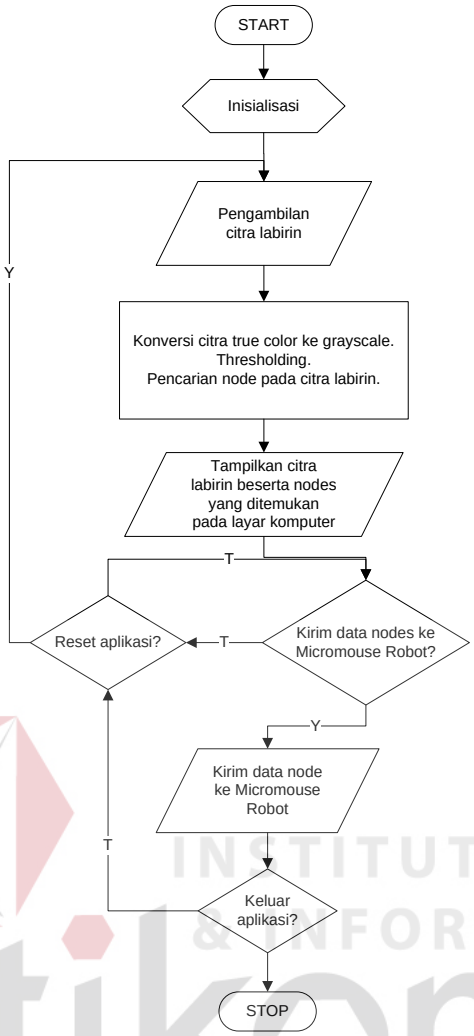



Gambar 3.10 (a) Bentuk *node* suatu persimpangan pada lintasan (b) Ilustrasi bentuk struktur data graf.

Gambar 3.10 (b) merupakan bentuk perwujudan bentuk struktur data dari *node* suatu persimpangan seperti yang ditunjukkan pada gambar 3.10 (a).

### 3.2.2 Perancangan Perangkat Lunak pada PC

Perancangan aplikasi pada PC menggunakan bahasa pemrograman Pascal pada compiler Borland Delphi 2010. *Flowchart* dari perancangan program secara umum seperti yang ditunjukkan pada gambar 3.11.




Gambar 3.11 *Flowchart* aplikasi pada PC.

Pada tahap awal PC mengambil citra dari area labirin dan mem-*filter*-nya menjadi hitam dan putih untuk mempermudah proses *scanning* area labirin tersebut. Proses *scanning* yang dimaksud yaitu memperoleh data *vertex*, menghitung jarak antara *vertex* satu ke *vertex* yang lain dan menyimpannya pada media penyimpanan dengan menggunakan struktur data graf dan DLL dari sebuah citra area labirin. Tiap *vertex* tetangga dari *vertex* referensi akan dimasukkan ke dalam sebuah struktur antrian data dan menandai *vertex* referensi menjadi sudah pernah dilalui agar tidak dilakukan pengecekan ulang. Setelah pengecekan sudah dilakukan semua terhadap *vertex* tetangga dari *vertex* referensi, maka *vertex*

referensi berikutnya adalah data yang dikeluarkan dari antrian data. Jika *vertex* keluaran dari antrian data tersebut sudah pernah dilalui maka antrian data akan mengeluarkan kembali data berikutnya. Proses *scanning* akan selesai jika pada antrian data tidak terdapat lagi data, dan melanjutkan ke proses penyelesaian masalah rute dengan jarak terpendek menggunakan algoritma Dijkstra. Data-data solusi jarak terpendek kemudian dikirimkan ke microcontroller pada *Micromouse Robot*.

## A. Perbaikan Citra

Pengambilan citra dilakukan dengan menggunakan media *webcam*, untuk mendapatkan penyebaran cahaya yang merata pada citra hasil pengambilan maka saat pengambilan citra diberi sumber cahaya tambahan. Pengambilan citra menggunakan Borland Delphi 2010 dengan media *webcam* dapat ditunjukkan pada penulisan program seperti berikut:

```
WCam.GetBitmap(FormMain.Img.Picture.Bitmap,nil,0);
```

Setelah citra sudah didapatkan maka dilakukan perbaikan citra yang bertujuan memudahkan dalam proses pencarian *node*. Perbaikan citra itu sendiri secara berurutan yaitu dengan perbaikan kecerahan, mengkonversi citra *true color* ke bentuk *grayscale*, dan kemudian mengkonversi kebentuk citra biner. Perbaikan kecerahan pada setiap *pixel* dapat ditunjukkan pada penulisan program seperti berikut:

```
P:=PCardinal(Cardinal(Line)+X*4);
PixColor:=P^;
R:=Integer((PixColor div $10000)and $FF)+Ptr.Brightness;
if R>$FF then
  R:=$FF
else if R<0 then
  R:=0;
G:=Integer((PixColor div $100)and $FF)+Ptr.Brightness;
```

```
if G>$FF then
   G:=$FF
else if G<0 then
   G:=0;
B:=Integer(PixColor and $FF)+Ptr.Brightness;
if B>$FF then
   B:=$FF
else if B<0 then
   B:=0;
PixColor:=(Cardinal(R)*$10000+Cardinal(G)*$100+Cardinal(B))and
$FFFFFF;
```

Konversi citra true color ke bentuk citra grayscale pada setiap *pixel* dapat ditunjukkan pada penulisan program seperti berikut:

```
P:=PCardinal(Cardinal(Line)+X*4);
Gray:=0;
Gray:=Gray+(P^div $10000 and $FF)*3 div 10;
Gray:=Gray+(P^div $100 and $FF)*59 div 100;
Gray:=Gray+(P^and $FF)*11 div 100;
P^:=Gray*$10000+Gray*$100+Gray;
```

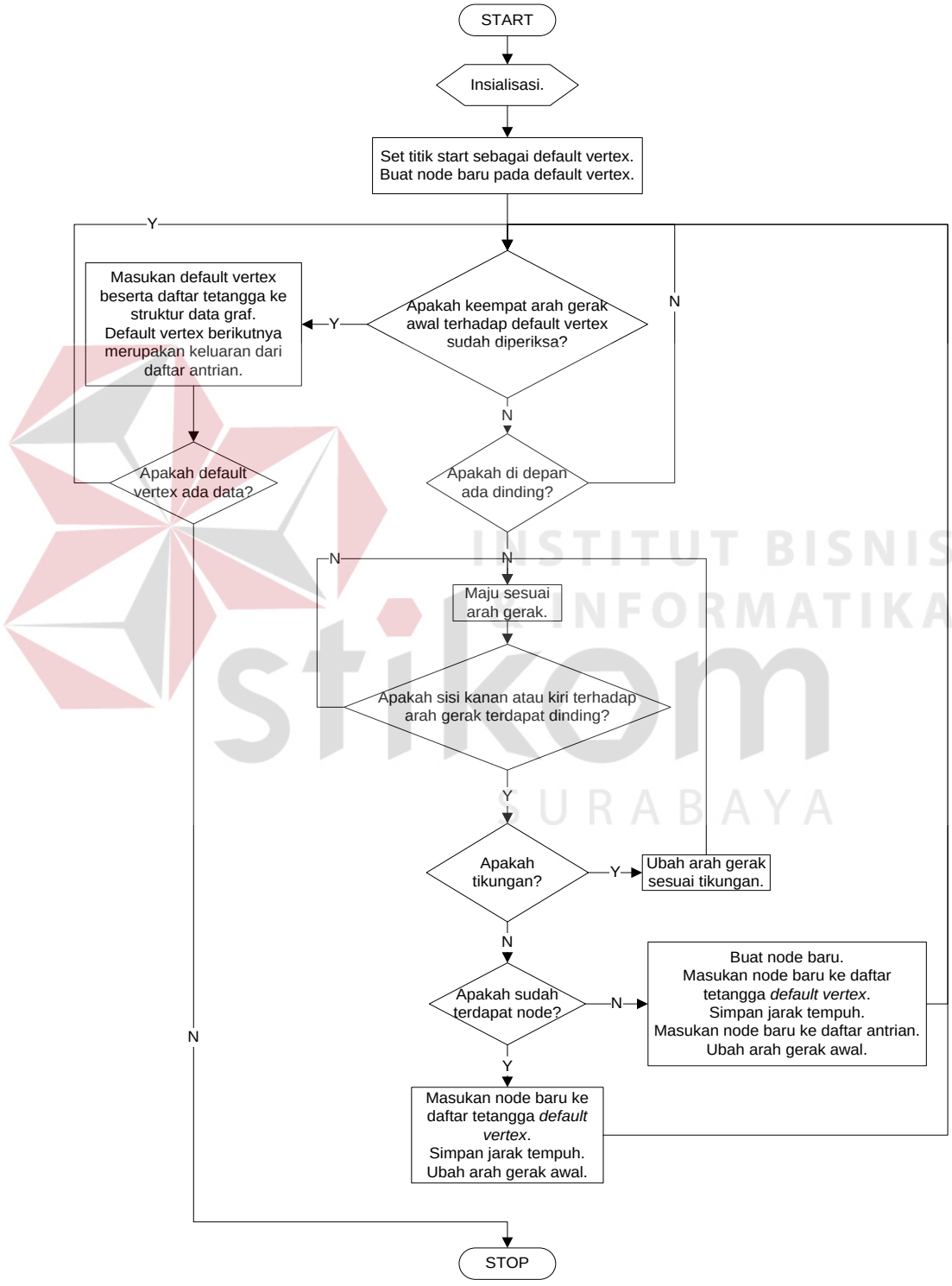
Sedangkan konversi kebentuk citra biner pada setiap *pixel* dapat ditunjukkan pada penulisan program seperti berikut:

```
if Integer(Gray and $FF)>(Ptr.Threshold) then
begin
   if InvertEn=False then
      PixColor:=$FFFFFF
   else
      PixColor:=0;
end
else
begin
   if InvertEn=False then
      PixColor:=0
   else
      PixColor:=$FFFFFF;
end;
```

Ptr.Threshold merupakan nilai ambang untuk menentukan nilai *pixel* yang baru. Pencarian *node* pada citra hanya dapat dilakukan pada citra labirin dengan lintasan berwarna putih dan dinding berwarna hitam. Jika pada labirin memiliki lintasan berwarna hitam dan dinding berwarna putih maka InvertEn memiliki fungsi untuk operasi *boolean* "*not*" sehingga didapat citra yang sesuai untuk pencarian *node*.

## B. Pencarian *Node*

Flowchart dari proses pencarian *node* pada citra labirin dapat ditunjukkan seperti pada gambar 3.12.




Gambar 3.12 *Flowchart* proses pencarian *node*.

Pencarian *node* awalnya dengan menentukan koordinat titik start sebagai titik pencarian awal dan menentukan nilai arah hadap proses pencarian. Arah hadap tersebut yang dimaksud yaitu 0 untuk utara, 1 untuk ke barat, 2 untuk ke selatan, dan 3 untuk ke timur pada citra yang diolah seperti arah mata angin pada peta secara umum. Dari titik awal, proses pencarian akan dilakukan pada empat arah hadap tersebut. Setelah menentukan arah hadap selanjutnya dilakukan pendeteksian adanya dinding pada sisi depan titik pencarian terhadap arah hadap. Pendeteksian dinding sisi depan titik pencarian terhadap arah hadap dapat ditunjukkan pada penulisan program seperti berikut:

```
Re:=False;
L:=Lbr;
for I:=0 to L do
begin
   P:=PCardinal(Integer(ImgBase)-
   ((_Data.Y+YCek[_Data.Arah,2]*I)*LineSize)+
   ((_Data.X+XCek[_Data.Arah,2]*I)*4));
   if P^=0 then {0 = hitam}
   begin
      Re:=True;
      Break;
   end;
end;
Result:=Re;
```

Lbr merupakan lebar lintasan dalam satuan *pixel*. Re merupakan *output* dari algoritma pendeteksian dinding sisi depan titik pencarian terhadap arah hadap yang dapat ditunjukkan seperti pada tabel 3.1.

Tabel 3.1 *Output* algoritma pendeteksian dinding sisi depan titik pencarian terhadap arah hadap.

| *Output* Re | Keterangan sisi depan |
|---|---|
| False | Tidak ada dinding |
| True | Ada dinding |

Jika pada sisi depan terdeteksi adanya dinding maka pencarian pada arah hadap tersebut akan dihentikan dan dilanjutkan pada arah hadap yang lain, jika tidak

tidak terdeteksi adanya dinding maka proses pencarian ke titik berikutnya akan dilanjutkan.

Untuk mendapatkan koordinat titik pencarian berikutnya dalam proses pencarian dapat dilakukan dengan menggunakan operasi penambahan antara titik pencarian saat ini dengan suatu nilai konstanta di dalam sebuah *array* dengan menggunakan *index* arah hadap yang dapat ditunjukkan pada penulisan program seperti berikut:

```
_Data.X:=_Data.X+XMove[_Data.Arah];
_Data.Y:=_Data.Y+YMove[_Data.Arah];
```

_Data.X dan _Data.Y merupakan koordinat atau titik pencarian pada citra. _Data.Length merupakan panjang jalur dalam satuan pixel yang sudah ditempuh, ditambahkan dengan 1 jika ada perubahan titik pencarian. _Data.Arah merupakan arah hadap. XMove dan Ymove merupakan suatu *constan array* yang dapat ditunjukkan pada pendeklarasian variabel seperti berikut:

```
const XMove: array [0..3] of Shortint=(0,-1,0,1);
const YMove: array [0..3] of Shortint=(-1,0,1,0);
```

Jarak merupakan komponen terpenting yang diolah pada penerapan algoritma Dijkstra. Aplikasi yang dibuat, jarak dihitung menggunakan operasi *pixel* pada citra labirin. Setiap iterasi pada proses perpindahan titik pencarian, maka jarak yang disimpan akan ditambah dengan 1, seperti yang ditunjukkan pada penulisan program seperti berikut:

```
Inc(NData.Length);
```

Jarak antara suatu *node* dengan *node* yang lain terhitung dan disimpan jika telah ditemukan *node* yang terhubung dengan *base node* seperti yang ditunjukkan pada penulisan program sebagai berikut:

```
Edge^:=NData.Length;
```

Dalam proses pencarian, adanya tikungan atau persimpangan akan dideteksi pada setiap perubahan titik pencarian. Adanya tikungan dapat dideteksi jika salah satu di sisi kanan atau kiri titik pencarian terhadap arah hadap tidak mendeteksi adanya dinding dan di sisi depan titik pencarian mendeteksi adanya dinding. Adanya persimpangan dapat dideteksi jika lebih dari satu sisi titik pencarian terhadap arah hadap yang tidak mendeteksi adanya dinding. Pendeteksian dinding dapat dilakukan dengan melakukan komparasi hasil pembacaan warna pada titik koordinat pada citra, jika mendeteksi warna hitam maka titik tersebut merupakan dinding. Algoritma pendeteksian dinding sisi kiri dan kanan titik pencarian terhadap arah hadap dapat ditunjukkan pada penulisan program seperti berikut:

```
L = Lbr;
A:=1;
F:=0;
for I:=0 to 1 do
begin
   for J:=0 to L do
   begin
      P:=PCardinal(Integer(ImgBase)-
      ((_Data.Y+YCek[_Data.Arah,I]*J)*LineSize)+
      ((_Data.X+XCek[_Data.Arah,I]*J)*4));
      {mendeteksi dinding}
      if P^and $FFFFFF=0 then {0 = hitam}
      begin
         F:=F or A;
         Break;
      end;
   end;
   A:=A*2;
end;
Result:=F;
```

XCek dan YCek merupakan suatu *constan array* yang didalam dapat ditunjukkan pada pendeklarasian variabel seperti berikut:

```
const XCek: array [0..3,0..2] of Shortint = ((-1,1,0), (0,0,-1),
(1,-1,0), (0,0,1));
const YCek: array [0..3,0..2] of Shortint = ((0,0,-1), (1,-1,0),
(0,0,1), (-1,1,0));
```

F merupakan *output* dari algoritma pendeteksian dinding sisi kiri dan kanan titik pencarian terhadap arah hadap yang dapat ditunjukkan seperti pada tabel 3.2.

Tabel 3.2 *Output* algoritma pendeteksian dinding sisi kiri dan kanan titik pencarian terhadap arah hadap.

| *Output* F | Keterangan sisi kiri | Keterangan sisi kanan |
|---|---|---|
| 0 | Tidak ada dinding | Tidak ada dinding |
| 1 | Tidak ada dinding | Ada dinding |
| 2 | Ada dinding | Tidak ada dinding |
| 3 | Ada dinding | Ada dinding |

Dari hasil pendeteksian dinding sisi depan, kiri, dan kanan titik pencarian terhadap arah hadap dapat diketahui adanya tikungan atau simpangan yang dapat ditunjukkan seperti pada tabel 3.3.

Tabel 3.3 Pendeteksian tikungan atau simpangan.

| Re | F | Sisi depan | Sisi kiri | Sisi kanan | Keterangan |
|---|---|---|---|---|---|
| False | 0 | Tidak ada dinding | Tidak ada dinding | Tidak ada dinding | Simpang empat |
| False | 1 | Tidak ada dinding | Tidak ada dinding | Ada dinding | Simpang tiga |
| False | 2 | Tidak ada dinding | Ada dinding | Tidak ada dinding | Simpang tiga |
| False | 3 | Tidak ada dinding | Ada dinding | Ada dinding | Jalur lurus |
| True | 0 | Ada dinding | Tidak ada dinding | Tidak ada dinding | Simpang tiga |
| True | 1 | Ada dinding | Tidak ada dinding | Ada dinding | Tikungan ke kanan |
| True | 2 | Ada dinding | Ada dinding | Tidak ada dinding | Tikungan ke kiri |
| True | 3 | Ada dinding | Ada dinding | Ada dinding | Jalan buntu |

Terdeteksinya tikungan ke kiri atau kanan dalam pencarian node maka arah hadap harus diubah sesuai dengan arah tikungan yang terdeteksi. Algoritma perubahan arah hadap proses pencarian *node* dapat ditunjukkan pada penulisan program seperti berikut:

```
if FBranch=2 then
```

```
  NData.Arah:=(NData.Arah+1)mod 4
else if FBranch=1 then
begin
  if NData.Arah=0 then
    NData.Arah:=4;
  Dec(NData.Arah);
end;
```

Algoritma tersebut dijalankan jika mendeteksi adanya dinding pada sisi depan. Nilai pada FBranch berasal dari output algoritma pendeteksian dinding sisi kiri dan kanan.

Dalam proses pencarian *node*, jika terdeteksi adanya persimpangan atau jalan buntu yang merupakan sebuah *node* maka selanjutnya akan ditentukan posisi dari titik tersebut. Posisi tersebut berada di tengah-tengah simpangan yang diperoleh dengan operasi penambahan titik pencarian saat ini dengan setengah kali lebar jalur yang dapat ditunjukkan seperti pada penulisan program sebagai berikut:

```
NData.X:=NData.X+XMove[NData.Arah]*(L div 2);
NData.Y:=NData.Y+YMove[NData.Arah]*(L div 2);
NData.Length:=NData.Length+(L div 2);
```

Ndata.X dan Ndata.Y merupakan posisi koordinat X dan Y titik pencarian yang kemudian bergeser ke sisi depan arah hadap sejauh L yang merupakan lebar lintasan dibagi dua. Pergeseran posisi titik pencarian saat ini diikuti dengan perubahan jarak melalui operasi penambahan jarak saat ini dengan lebar lintasan dibagi dua.

Dalam proses pencarian *node*, jika terdeteksi adanya persimpangan atau jalan buntu yang merupakan sebuah *node* maka akan dimasukkan ke dalam daftar tetangga titik awal pencarian yang dapat ditunjukkan pada penulisan program seperti berikut:

```
if Matrix[NData.X,NData.Y]=nil then
begin
  MCreate(@NData^);
```

```
   Inc(NData.ID^);
   New(Edge);
   Edge^:=NData.Length;
   ArTmp:=NData.Arah;
   NData.Arah:=ArDef;
   NData.List.DataIn(@NnodeCreate(@Matrix[NData.X,NData.Y]^,
   @Edge^,@NData^)^);
   NData.Arah:=ArTmp;
end
else
begin
   ArTmp:=NData.Arah;
   NData.Arah:=ArDef;
   NNodeCek(@NData^);
   NData.Arah:=ArTmp;
end;
```

Jika *node* tersebut merupakan *node* baru yang berarti simpangan atau jalan buntu yang belum pernah menjadi titik awal maka *node* tersebut akan dimasukkan dalam struktur data antrian. Untuk menandai sebuah node pernah menjadi titik awal atau belum dapat ditunjukkan pada penulisan program seperti berikut:

```
Data.MPtr.NeighborList:=@Data.List^;
Data.MPtr.Status:=True;
Graph.DataIn(@Data.MPtr^);
I:=0;
repeat
   InfoNOde:=Data.List.DataPointer[I];
   if InfoNode<>nil then
   begin
      if InfoNode.Vertex.Status=False then
      Queue.DataIn(@InfoNode.Vertex^);
   end;
   Inc(I);
until (InfoNode=nil);
```

Sebuah titik yang saat ini menjadi titik awal pada statusnya akan dibuat menjadi *true* dan dimasukkan pada struktur data Graph. Pada *node* tetangga titik awal yang memiliki status *false* akan dimasukkan pada struktur data antrian.

Jika keempat arah dari titik awal sudah dilakukan pencarian node, untuk titik awal berikutnya merupakan *output* dari antrian yang dapat ditunjukkan pada penulisan program seperti berikut:

```
repeat
   Data.MPtr:=@Queue.DataOut^;
```

```
  if Data.MPtr<>nil then
  begin
    if Data.MPtr.Status=False then Break;
  end
  else
    Break;
until (True=False);
```

Setelah didapatkan titik awal berikutnya dari antrian maka titik tersebut akan diperika kembali statusnya. Jika titik awal tersebut memiliki status *false* maka proses pencarian *node* akan dilakukan kembali pada keempat arah dari titik hadap. Proses pencarian *node* akan berhenti secara keseluruhan jika tidak ditemukan lagi data titik pada antrian.

### C. Pengiriman Data Graf ke Mikrokontroler

Pengiriman data graf ke mikrokontroler dilakukan dengan mengirim karakter FF heksadesimal untuk pertama kali seperti yang ditunjukkan pada penulisan program seperti berikut:

```
FormMain.Comport.Write($FF,1);
```

Mikrokontroler akan menerima dan mengirim karakter FF heksadesimal pada PC jika siap melakukan proses penerimaan data yaitu jumlah *node* yang terdeteksi pada citra yang dapat ditunjukkan pada penulisan program seperti berikut:

```
FormMain.Comport.Write(Graph.Count,1);
```

Setelah menerima dan siap untuk data berikutnya maka mikrokontroler mengirim karakter FF heksadesimal.

Data berikutnya yang dikirim ke mikrokontroler yaitu data *vertex* dan *edge* yang dapat ditunjukkan pada penulisan program seperti berikut:

```
for I:=0 to Graph.Count-1 do
begin
  Vtx:=@Graph.DataPointer[I]^;
  f_uart := false;
  for J:=0 to 3 do
```

```
  begin
    F:=False;
    for K:=0 to Vtx.NeighborList.Count-1 do
    begin
      N:=@Vtx.Neighbor[K]^;
      if N.Arah=Shortint(J) then
      begin
        F:=True;
        SerialData := byte(N.Edge^ div 256);
        FormMain.Comport.Write(SerialData,1);
        SerialData := byte(N.Edge^ and $FF);
        FormMain.Comport.Write(SerialData,1);
        Break;
      end;
    end;
    if f = false then
    begin
      SerialData := $FF;
      FormMain.Comport.Write(SerialData,1);
      FormMain.Comport.Write(SerialData,1);
    end;
  end;
  while f_uart = false do;
  f_uart := false;
  for J:=0 to 3 do
  begin
    F:=False;
    for K:=0 to Vtx.NeighborList.Count-1 do
    begin
      N:=@Vtx.Neighbor[K]^;
      if N.Arah=Shortint(J) then
      begin
        F:=True;
        SerialData := byte(N.Vertex.ID);
        Break;
      end;
    end;
    if f = false then
    begin
      SerialData := $FF;
    end;
    FormMain.Comport.Write(SerialData,1);
  end;
  while f_uart = false do;
end;
```

Pengiriman dilakukan secara berurutan, dari data *edge* kemudian data *vertex*, dari *index* ke-0 hingga ke-jumlah *node* dikurangi 1 pada graf dan setiap *node* tetangga dikirim secara berurutan, dari arah hadap 0 hingga 3. Jika tidak ditemukan *node* tetangga pada suatu arah hadap maka *vertex* yang dikirim bernilai FF heksadesimal dan *edge* yang dikirim bernilai FFFF heksadesimal. PC akan

menunggu karakter FF heksadesimal dari mikrokontroler setiap kali selesai mengirimkan satu set data *edge* dan *vertex* sebesar 12 Byte dan mengirim data *edge* dan *vertex* berikutnya jika sudah menerima karakter tersebut.

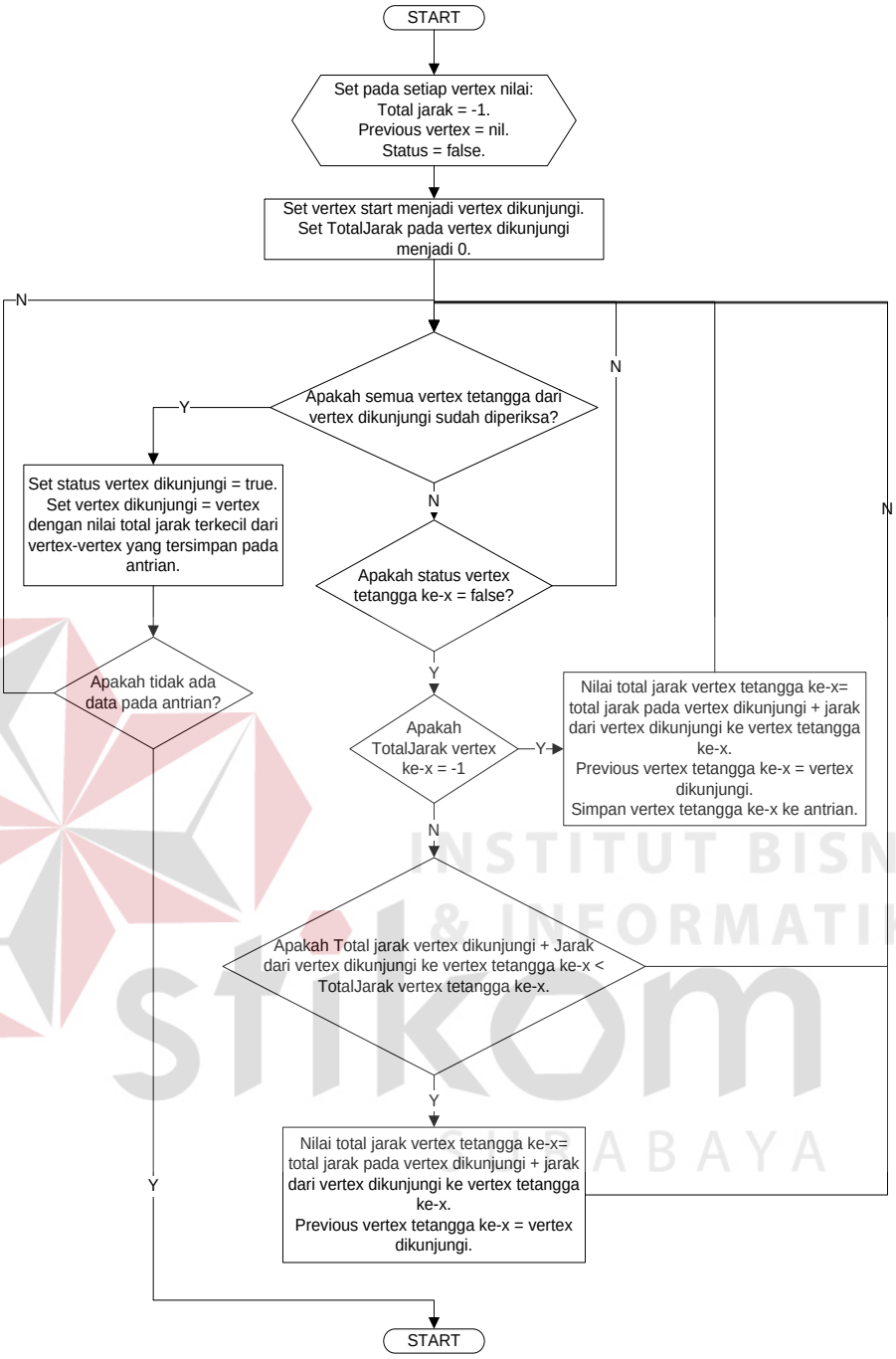
Pengiriman data berikutnya yaitu data *start node*, *finish node*, dan arah hadap pada posisi *start* dan diakhiri dengan menerima karakter FF heksadesimal dari mikrokontroler yang dapat ditunjukkan pada penulisan program seperti berikut:

```
f_uart := false;
Vtx := @matrix[xStart,yStart]^;
SerialData := vtx.ID;
FormMain.Comport.Write(SerialData,1);
Vtx := @matrix[xFinish,yFinish]^;
SerialData := vtx.ID;
FormMain.Comport.Write(SerialData,1);
Vtx := @matrix[xFinish,yFinish]^;

Vtx:=@matrix[xStart, yStart]^;
for K := 0 to Vtx.NeighborList.Count- 1 do
begin
  N:=@Vtx.Neighbor[K]^;
  if @N.Vertex.Previous^ = @Vtx^ then
  begin
    FormMain.Comport.Write(byte(N.Arah),1);
    while f_uart = false do;
    break;
  end;
end;
```

### 3.2.3 Penerapan Algoritma Dijkstra

*Flowchart* dari algoritma Dijkstra dapat ditunjukkan seperti pada gambar 3.13.




Gambar 3.13 *Flowchart* algoritma Dijkstra.

Penerapan algoritma Dijkstra dimulai dengan proses inisialisasi, pada PC dapat ditunjukkan pada penulisan program seperti berikut:

```
{Inisialisasi}
for A:=0 to Graph.Count-1 do
begin
  GP:=PVertex(Graph.DataPointer[A]);
  GP.Weight:=-1;
```

```
   GP.Previous:=nil;
   GP.Status:=True;
end;
```

Sedangkan pada Mikrokontroler dapat ditunjukkan pada penulisan program seperti berikut:

```
struct vtx
{
   u16 weight;
   u8 previous, status;
} vertex[254];
for (i = 0; i < 254; i++)
{
   vertex[i].weight = 0xFFFF;
   vertex[i].previous = 0xFF;
   vertex[i].status = 0;
   q_node[i] = 0xFF;
   dir[i] = 0xFF;
}
```

Proses inisialisai tersebut dilakukan pada keseluruhan *node* memberi nilai -1 dan FFFF heksadesimal pada *weight* yang digunakan untuk total jarak dari *start node* ke masing-masing *node*, memberi nilai *nil* dan FF heksadesimal pada *previous* yang digunakan sebagai penunjuk *node* tetangga dengan jarak terpendek, dan memberi nilai true dan 0 pada *status* yang digunakan sebagai penanda sudah atau tidaknya suatu *node* dikunjungi.

Setelah proses inisialisasi maka menunjuk *start node* sebagai *base node* dan memberi *weight* pada *start node* dengan nilai nol (0), pada PC ditunjukkan pada penulisan program seperti berikut:

```
GPointer:=@Matrix[XStart,YStart]^;
GPointer.Weight:=0;
```

Sedangkan pada Mikrokontroler dapat ditunjukkan pada penulisan program seperti berikut:

```
gpointer = start_node;
vertex[gpointer].weight = 0;
```

GPointer merupakan *base node*.

Pada masing-masing *node* yang merupakan tetangga dari *base pointer* dilakukan komparasi pada status, jika status bernilai *true* yang berarti belum pernah dikunjungi maka selanjutnya dilakukan komparasi terhadap *weight*. Jika *weight* bernilai -1 maka *weight* dari *node* tetangga tersebut diberi nilai hasil penambahan antara nilai *weight* dari *base node* dengan nilai *edge* pada *node* tetangga, *previous* menunjuk pada *base node*, dan *node* tetangga tersebut dimasukkan pada struktur data *ascending* yaitu struktur data yang mengurutkan nilai *weight* dari yang terkecil hingga terbesar, jika nilai *weight* tidak sama dengan -1 maka *weight* diberi nilai terkecil antara nilai *weight* dengan nilai hasil penambahan antara nilai *weight* dari *base node* dengan nilai *edge* pada *node* tetangga. Setelah masing-masing *weight node* tetangga dari *base pointer* tidak ada yang bernilai negatif maka status *base node* diberi nilai *false* dan *base node* selanjutnya merupakan *output* dari struktur data *ascending* yaitu *node* yang memiliki nilai *weight* terkecil pada struktur penyimpanan data tersebut. Proses-proses tersebut pada PC dapat ditunjukkan pada penulisan program seperti berikut:

```
{Dijkstra}
repeat
  A:=0;
  Cnt:=GPointer.NeighborList.Count;
  while A<Cnt do
  begin
    Vtx:=@GPointer.NeighborList.DataPointer[A]^;
    if Vtx.Vertex.Status=True then
    begin
      if Vtx.Vertex.Weight=-1 then
      begin
        Vtx.Vertex.Weight:=GPointer.Weight+Vtx.Edge^;
        Vtx.Vertex.Previous:=@GPointer^;
        VQ.DataIn(@Vtx.Vertex^);
      end
      else
      begin
        if GPointer.Weight+Vtx.Edge^<Vtx.Vertex.Weight then
        begin
          Vtx.Vertex.Weight:=GPointer.Weight+Vtx.Edge^;
```

```
            Vtx.Vertex.Previous:=@GPointer^;
          end;
        end;
      end;
      Inc(A);
    end;
    GPointer.Status:=False;
    GP:=VQ.DataOut;
    if GP<>nil then GPointer:=@GP^;
until (GP=nil);
```

Sedangkan pada mikrokontroler dapat ditunjukkan pada penulisan program seperti berikut:

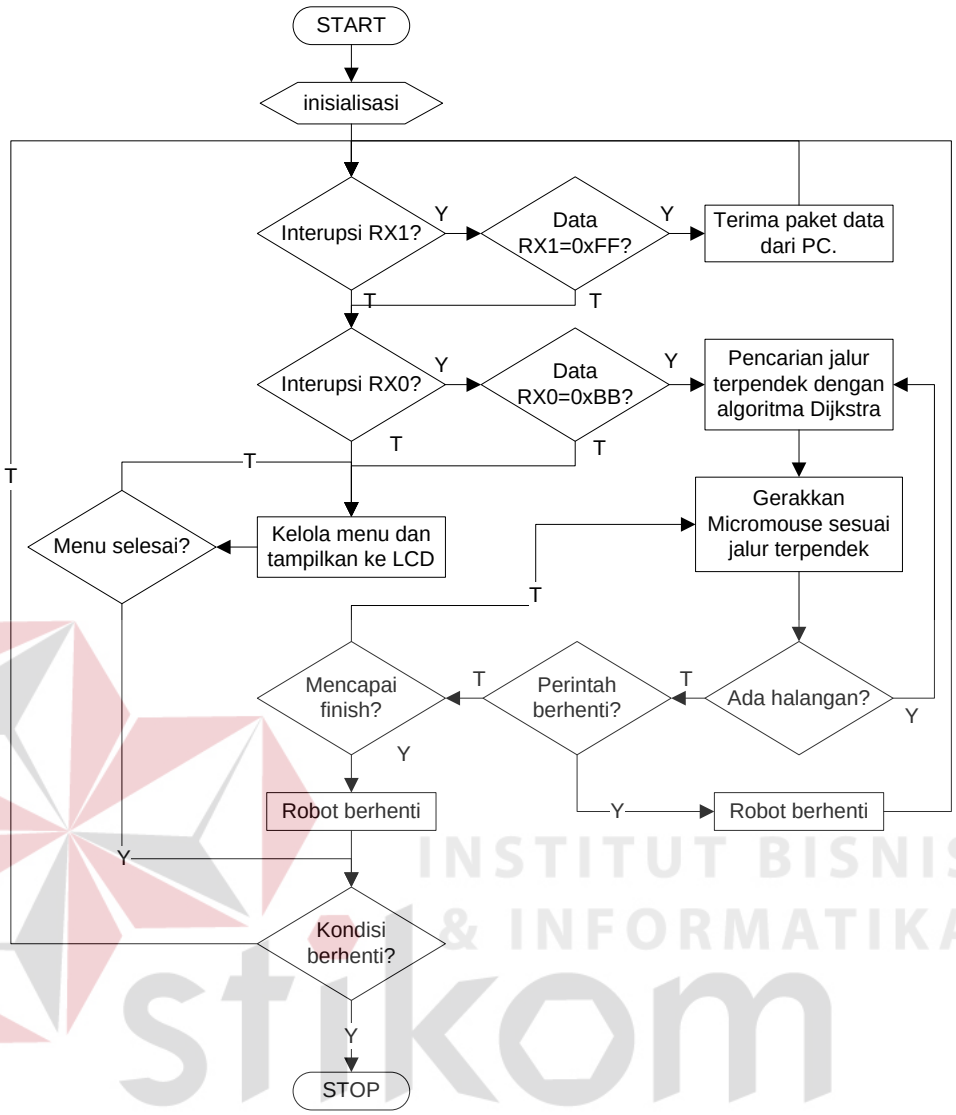
```
q_count = 0;
do
{
   putchar1(gpointer);
   for (i = 0; i<4; i++)
   {
      putchar(11);
      putchar(gpointer);
      putchar(i);
      n = getchar();
      if (n == 0xFF) continue;
      if (vertex[n].status == 0)
      {
         if (vertex[n].weight == 0xFFFF)
         {
            vertex[n].weight=vertex[gpointer].weight+
            edge[gpointer].data_edge[i];
            vertex[n].previous = gpointer;
            q_node[q_count] = n; q_count++;
         }
         else
         {
            if(vertex[gpointer].weight + edge[gpointer].data_edge[i]
            < vertex[n].weight)
            {
               vertex[n].weight=vertex[gpointer].weight+
               edge[gpointer].data_edge[i];
               vertex[n].previous = gpointer;
            }
         }
      }
   }
   vertex[gpointer].status = 1;
   q_index = 0;
   gpointer = q_node[0];
   for (i = 1; i < q_count; i++)
   {
      if (vertex[q_node[i]].weight < vertex[gpointer].weight)
      {
         gpointer = q_node[i];
         q_index = i;
      }
   }
```

```
    for (i = q_index+1; i < q_count; i++) q_node[i-1] = q_node[i];
    if (q_count != 0) q_count--;
    q_node[q_count] = 0xFF;
}
while (gpointer != 0xFF);
```

Proses pencarian rute terpendek akan berhenti jika pada struktur data *ascending* tidak terdapat data *node* pada struktur data tersebut dan rute terpendek dapat diketahui dari dari rangkaian yang dibentuk oleh *previous*.

### 3.2.4 Perancangan Perangkat Lunak pada Mikrokontroler ATMega644P

Perancangan perangkat lunak pada mikrokontroler menggunakan bahasa pemrograman C pada compiler Code Vision AVR 2.03.4. *Flowchart* dari perancangan program secara umum seperti yang ditunjukkan pada gambar 3.14.




Gambar 3.14 *Flowchart* program pada mikrokontroler ATMega644P

Pada tahap awal mikrokontroler akan menerima data solusi dari permasalahan pencarian rute dengan jarak terpendek dari titik *start* ke *finish* yang dikirim oleh PC dan meletakkan data solusi tersebut dalam antrian data. Setelah data solusi sudah diterima maka *Micromouse Robot* siap melakukan perjalanan menuju ke *finish*. Selama perjalanan robot akan mengecek tiap *sensor* yang terpasang untuk mendeteksi jarak dinding dan ada atau tidaknya persimpangan. Jika terdapat persimpangan maka referensi arah belok berasal dari data yang

dikeluarkan dari antrian data dan apabila telah mencapai *finish* maka *robot* akan berhenti.

## A. Komunikasi Data dengan PC

Untuk berkomunikasi dengan PC, pada ATMega644P menggunakan protokol *Universal Asynchronous Receiver-Transmitter* (UART) pada *special function* USART1 dengan parameter yang ditunjukkan seperti pada tabel 3.4.

Tabel 3.4 Parameter komunikasi data dengan PC menggunakan protokol UART.

| Parameter | Keterangan |
|---|---|
| Data bit | 8 bit data, 1 *stop* bit, tanpa *parity* |
| *Receiver* | Aktif |
| *Transmitter* | Aktif |
| *Mode* | Asinkron |
| *Baudrate* | 115200 bps (*double speed mode*) |

Terdapat register 16 bit untuk mengatur baudrate yaitu UBRR1. Untuk mendapat baudrate 115200 bps maka nilai register UBRR1 dapat dihitung seperti berikut:

$$UBRR1 = \frac{11059200}{8 \times 115200} - 1$$

$$UBRR1 = 000B\ heksadesimal$$

Untuk menjalankan *spesial function* USART1 sesuai dengan parameter pada tabel maka *register* diatur seperti berikut:

```
UCSR1A=0x02;
UCSR1B=0x98;
UCSR1C=0x06;
UBRR1H=0x00;
UBRR1L=0x0B;
```

## B. Komunikasi Data dengan ATMega328P.

Untuk berkomunikasi dengan ATMega328P, pada ATMega644P menggunakan protokol UART pada *special function* USART0 dengan parameter yang ditunjukkan seperti pada tabel 3.5.

Tabel 3.5 Parameter komunikasi data dengan ATMega328P menggunakan protokol UART.

| Parameter | Keterangan |
|---|---|
| Data bit | 8 bit data, 1 *stop* bit, tanpa *parity* |
| *Receiver* | Aktif |
| *Transmitter* | Aktif |
| *Mode* | Asinkron |
| *Baudrate* | 1382400bps (*double speed mode*) |

Untuk mendapat baudrate 1382400 bps maka nilai register UBRR0 dapat dihitung seperti berikut:

$$UBRR0 = \frac{11059200}{8 \times 1382400} - 1$$

$$UBRR0 = 0000\ heksadesimal$$

Untuk menjalankan *spesial function* USART0 sesuai dengan parameter pada tabel maka *register* diatur seperti berikut:

```
UCSR0A=0x02;
UCSR0B=0x98;
UCSR0C=0x06;
UBRR0H=0x00;
UBRR0L=0x00;
```

## C. Penerimaan Data Graf dari PC

Penyimpanan data graf dibagi menjadi dua bagian karena adanya keterbatasan kapasitas pada EEPROM mikrokontroler, pada EEPROM ATMega644P menyimpan data edge dan EEPROM ATMega328P menyimpan data vertex. Penyimpanan pada EEPROM bertujuan agar data tidak hilang saat

tidak ada arus catu daya yang mengalir pada kedua mikrokontroler sehingga data tersebut dapat digunakan lagi. Struktur data untuk menyimpan data edge pada EEPROM dapat ditunjukkan pada penulisan program seperti berikut:

```
eeprom struct dt
{
    u16 data_edge[4];
}   edge[254];
```

Struktur data yang telah dibuat hanya dapat menampung kurang dari sama dengan 254 *node* dengan ukuran memory yang dibutuhkan sebesar 2 Byte × 4 × 254 yaitu 2032 byte.

Proses penerimaan data graf dimulai saat mikrokontroler menerima karakter FF heksadesimal dan dilanjutkan dengan mengirim karakter FF heksadesimal ke PC sebagai tanda siap menerima data yang dapat ditunjukkan pada penulisan program seperti berikut:

```
putchar1(0xFF);
```

Kemudian menerima data jumlah *node* yang sudah terdeteksi pada citra dari PC yang dapat ditunjukkan pada penulisan program seperti berikut:

```
node_count = getchar1();
```

Setelah data jumlah *node* diterima maka dikirimkan kembali karakter FF heksadesimal sebagai tanda siap menerima data berikutnya yaitu data *vertex* dan *edge* yang dapat ditunjukkan pada penulisan program seperti berikut:

```
putchar1(0xFF);
for (i = 0; i < node_count; i++)
{
   for (j = 0; j < 4; j++)
   {
   edge[i].data_edge[j] = (u16)getchar1() * 256;
   edge[i].data_edge[j] |= getchar1();
   }
   putchar1(0xFF);
   for (j = 0; j < 4; j++)
   {
     putchar(10);
     putchar(i);
```

```
      putchar(j);
      putchar(getchar1());
      getchar();
   }
   putchar1(0xFF);
}
```

ATMega644P menerima data sebesar 12 Byte yang terbagi menjadi 8 Byte data *edge* dan 4 Byte data *vertex*. 8 Byte data *edge* yang diterima disimpan pada EEPROM ATMega328P. 4 Byte data *vertex* yang diterima, dikirim dan disimpan pada EEPROM ATMega328P. Setelah menerima data *vertex* dan *edge*, maka siap menerima data selanjutnya dari PC yaitu data *start node*, *finish node*, dan arah hadap awal pada posisi *start* yang dapat ditunjukkan pada penulisan program seperti berikut:

```
start_node = getchar1();
finish_node = getchar1();
d_dir = getchar1();
putchar1(0xFF);
```

yang diakhiri dengan mengirim karakter FF heksadesimal ke PC sebagai penanda diterimanya semua data dan berakhirnya penerimaan data.



**D. Pembacaan Data Sensor Reflektansi IR**

Sensor reflektansi IR yang yang digunakan untuk mengukur jarak dinding dengan *Mikromouse Robot* berjumlah 5 buah yang masing-masing terhubung dengan *port* ADC0-ADC4. Agar *special function* ADC pada ATMega644P dapat digunakan maka perlu mengatur beberapa register seperti berikut:

```
DIDR0=0x1F;
ADMUX=0x20;
ADCSRA=0x88;
```

Dari register tersebut menghasilkan *special function* ADC dengan parameter yang dapat ditunjukkan seperti pada tabel 3.6.

Tabel 3.6 Parameter ADC.

| Parameter | Keterangan |
|---|---|
| ADC *enable* | Aktif |
| Data bit | 8 bit |
| *Interrupt* | Aktif |
| *Noise Canceler* | Aktif |
| *Volt Reference* | Pin VREF |
| *Clock* | 5,5296 MHz |
| *Disable digital input buffer* | ADC0 – ADC4 |

Setelah register sudah diatur maka *special function* ADC dapat diakses yang ditunjukkan pada penulisan program seperti berikut:

```
for (i = 0; i < 5; i++)
  sens[i] = (read_adc(i));
```

sens[i] merupakan sebuah variabel yang menyimpan nilai yang terbaca dari masing-masing *port* ADC. Jika jarak dinding semakin jauh maka nilai yang terbaca akan semakin kecil dan semakin besar nilai yang terbaca jika jarak dinding semakin dekat, dengan lebar antara 0-255.

### 3.2.5 Perancangan Perangkat Lunak pada ATMega328P

**A. Komunikasi Data dengan ATMega644P**

Untuk berkomunikasi dengan ATMega644P, pada ATMega328P menggunakan protokol UART pada *special function* USART0 dengan parameter yang ditunjukkan seperti pada tabel 3.7.

Tabel 3.7 Parameter komunikasi data dengan PC menggunakan protokol UART.

| Parameter | Keterangan |
|---|---|
| Data bit | 8 bit data, 1 *stop* bit, tanpa *parity* |
| *Receiver* | Aktif |
| *Transmitter* | Aktif |
| *Mode* | Asinkron |
| *Baudrate* | 1382400bps (*double speed mode*) |

Untuk mendapat baudrate 1382400 bps maka nilai register UBRR0 dapat dihitung seperti berikut:

$$UBRR0 = \frac{11059200}{8 \times 1382400} - 1$$

$$UBRR0 = 0000\ heksadesimal$$

Untuk menjalankan *spesial function* USART0 sesuai dengan parameter pada tabel maka *register* diatur seperti berikut:

```
UCSR0A=0x02;
UCSR0B=0x98;
UCSR0C=0x06;
UBRR0H=0x00;
UBRR0L=0x00;
```



**B. Penerimaan Data *Vertex* dari ATMega644P**

Pada ATMega328P memiliki struktur data yang digunakan untuk menyimpan data *vertex* yang penyimpanannya dilakukan ke dalam EEPROM. Struktur data *vertex* pada ATMega328P dapat ditunjukkan seperti berikut:

```
eeprom struct dt
{
  u8 data_node[4];
}node[254];
```

Struktur data yang telah dibuat hanya dapat menampung kurang dari sama dengan 254 *node* dengan ukuran memory yang dibutuhkan sebesar 1 Byte × 4 × 254 yaitu 1016 Byte.

ATMega328P akan menyimpan data vertex sesuai dengan instruksi dari ATMega644P yang dapat ditunjukan pada penulisan program seperti berikut:

```
case 10:
   while(rx_counter0 == 0);
   data = getchar();
   while(rx_counter0 == 0);
   data2 = getchar();
   while(rx_counter0 == 0);
   node[data].data_node[data2] = getchar();
   putchar(0xFF);
   break;
```

Instruksi yang dikirim oleh yang dikirim oleh ATMega644P akan dikomparasi oleh ATMega328P, jika instruksi tersebut bernilai 10 desimal maka ATMega328P akan menunggu dan menerima 2 Byte data yang digunakan sebagai *index* pada struktur data dan 1 Byte merupakan data *vertex* tersebut. Setelah selesai melakukan proses penyimpanan maka diakhiri dengan mengirim karakter FF heksadesimal ke ATMega644P.

ATMega644P juga dapat melakukan instruksi meminta data vertex dari ATMega328P yang dapat ditunjukkan pada penulisan program seperti berikut:

```
case 11:
   while(rx_counter0 == 0);
   data = getchar();
   while(rx_counter0 == 0);
   data2 = getchar();
   putchar(node[data].data_node[data2]);
   break;
```

Instruksi yang dikirim oleh yang dikirim oleh ATMega644P akan dikomparasi oleh ATMega328P, jika instruksi tersebut bernilai 11 desimal maka ATMega328P akan menunggu dan menerima 2 Byte data yang digunakan sebagai *index* pada struktur data. Kemudian ATMega328P mengirim data yang berada pada *index* yang telah diterima untuk dikirim kembali ke ATMega644P.