BAB 5 PERANCANGAN DAN IMPLEMENTASI

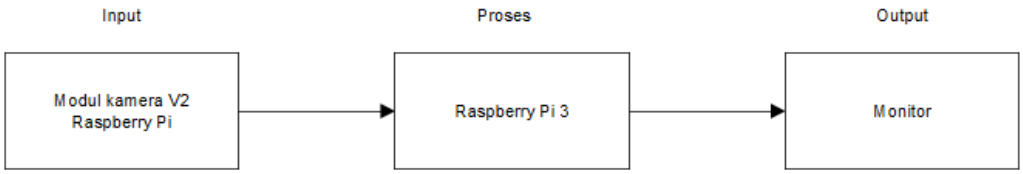
Pada bab ini akan membahas proses perancangan dari prototipe sistem yang akan dibuat mulai dari perangkat keras dan perangkat lunaknya, hingga proses implementasi dari prototipe pada tempat parkir.

5.1 Perancangan Sistem

Pada sub bab perancangan sistem akan menjelaskan diagram alir sistem dan proses-proses yang dilakukan pada sistem.

5.1.1 Diagram Blok Sistem

Setelah dilakukan rekayasa kebutuhan terkait kebutuhan dari sistem, maka dapat dibuat sebuah diagram blok sistem. Diagram blok menjelaskan cara kerja sistem secara keseluruhan. Mulai dari proses pengambilan input sistem hingga sistem dapat menghasilkan *output*. Diagram blok dari sistem adalah sebagai berikut:



Gambar 5.1 Diagram blok sistem

Pada diagram blok diatas dapat dilihat bahwa modul kamera akan mengambil citra tempat parkir untuk dijadikan sebagai *input* dari sistem. Selanjutnya citra yang telah diambil akan diproses pada raspberry pi 3 sehingga sistem dapat mengetahui jumlah dan ketersediaan *slot* parkir yang ada pada tempat parkir yang digunakan sebagai *input*. Kemudian hasil dari proses yang dilakukan pada raspberry pi 3 akan ditampilkan melalui monitor.

5.1.2 Perancangan Perangkat Keras

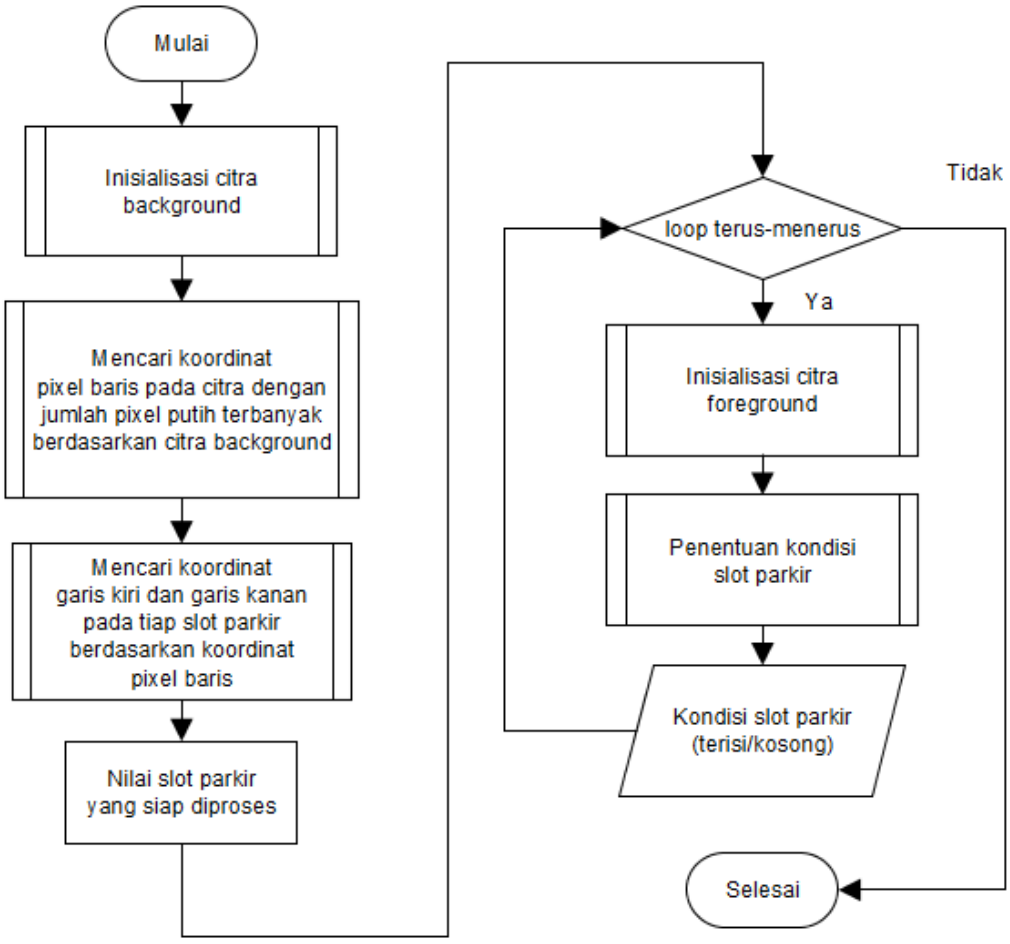
Perancangan perangkat keras merupakan tahapan dalam merancang prototipe sistem dimana digunakan 2 komponen utama yaitu modul kamera V2 raspberry pi dan raspberry pi 3.

Gambar 5.2 Perancangan perangkat keras sistem

Dari Gambar 5.2 dapat dilihat bahwa modul kamera V2 Raspberry Pi dihubungkan melalui *port* kamera yang tersedia pada Raspberry Pi 3. Apabila Raspberry Pi 3 dalam kondisi menyala, maka modul kamera V2 Raspberry Pi langsung dapat digunakan untuk mengambil ataupun merekam citra dengan menggunakan perintah *raspistill (*untuk mengambil citra) dan *raspivid* (untuk merekam citra) pada *command prompt* raspberry pi 3. Raspberry Pi 3 terhubung secara *remote* melalui *w*ifi dengan perangkat laptop sehingga seluruh proses dan hasil *output* dari sistem akan ditampilkan melalui perangkat laptop.

5.1.3 Perancangan Perangkat Lunak

Setelah prototipe sistem selesai dibuat, hal selanjutnya yang dilakukan adalah merancang perangkat lunak dari prototipe sistem berupa diagram alir dari program prototipe sistem secara keseluruhan. Secara garis besar, prototipe sistem akan bekerja sesuai diagram alir berikut:

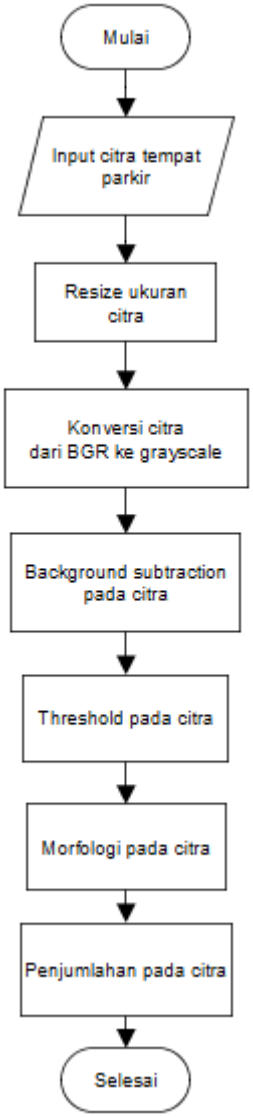


Gambar 5.3 Diagram alir sistem

5.1.3.1 Perancangan fungsi inisialisasi citra

Sistem memiliki beberapa fungsi, diantaranya yaitu fungsi inisialisasi citra, mencari koordinat *pixel* baris pada citra dengan jumlah *pixel* putih terbanyak

berdasarkan citra *background*, mencari koordinat garis kiri dan garis kanan pada tiap *slot* parkir berdasarkan koordinat *pixel* baris dan penentuan kondisi *slot* parkir. Fungsi inisialisasi citra terdiri dari beberapa proses yang dijelaskan pada Gambar 5.4 berikut.



Gambar 5.4 Diagram alir fungsi inisialisasi citra

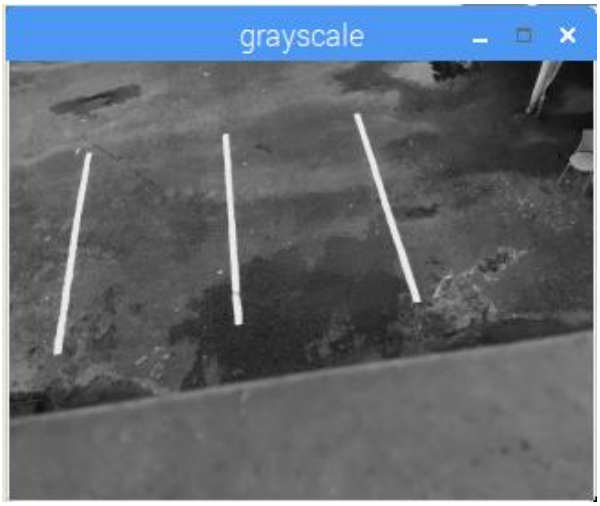
Tahapan pertama dari inisialisasi citra adalah mengambil input berupa citra dari tempat parkir tujuan. Untuk inisialisasi citra *background*, citra yang diambil adalah citra tempat parkir pada saat kondisi tempat parkir dalam keadaan belum terisi sama sekali, sedangkan untuk citra *foreground*, citra yang diambil adalah citra tempat parkir pada saat sistem dijalankan untuk mendeteksi ketersediaan tempat parkir dengan ukuran 640 x 480 *pixel*.

Gambar 5.5 *Input* citra tempat parkir yang digunakan sebagai citra *background*

Lalu citra yang telah diambil dan akan digunakan sebagai *input* akan di*resize* ukurannya menjadi 320 x 240 *pixel*. Hal ini dilakukan untuk mengurangi beban kinerja dari raspberry pi.

Gambar 5.6 *Input* citra yang telah di*resize*

Kemudian citra yang telah di*resize* akan dikonversi dari citra rgb menjadi citra *grayscale*. Citra yang awalnya terdiri dari intensitas warna merah, biru dan hijau akan dirubah sehingga hanya memiliki intensitas warna hitam, putih dan keabuan.



Gambar 5.7 *Input* citra dalam *color-space grayscale*

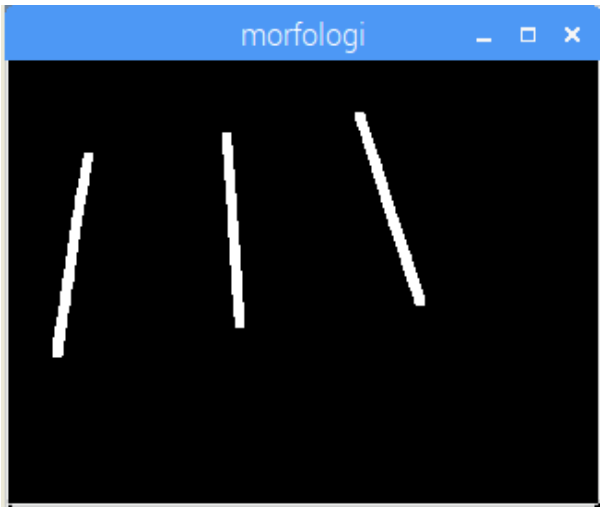
Selanjutnya dilakukan penerapan metode *background subtraction* dengan mengurangi citra *foreground* dengan citra *background* yang telah dikonversi menjadi *grayscale*.

Gambar 5.8 Citra hasil penerapan metode *background subtraction*

Setelah itu citra akan dirubah menjadi citra biner dan dilakukan proses *threshold*. Hal ini dilakukan agar citra hanya memiliki dua intensitas nilai, yaitu intensitas putih dengan nilai 255 dan intensitas hitam dengan nilai 0 berdasarkan nilai ambang yang diberikan. Apabila nilai suatu *pixel* kurang dari nilai ambang yang diberikan, maka nilai *pixel* akan diubah menjadi intensitas hitam, sedangkan apabila nilai *pixel* lebih dari atau sama dengan nilai ambang yang diberikan, maka nilai *pixel* akan diubah menjadi intensitas putih.

Gambar 5.9 Citra yang telah diberikan *threshold*

Langkah selanjutnya adalah melakukan operasi morfologi pada citra yang telah diberikan *threshold* yang terdiri dari 2 proses, yaitu *opening* dan dilasi, dimana operasi *opening* merupakan proses yang diawali dengan operasi dilasi dan kemudian diikuti dengan operasi erosi. Operasi *opening* dilakukan untuk menghilangkan *noise* berupa titik-titik putih kecil yang dapat menyebabkan sistem mengalami *error* dalam mendeteksi ketersediaan *slot* parkir. Kemudian operasi morfologi dilanjutkan dengan melakukan operasi dilasi yang bertujuan untuk menebalkan *objek* yang ingin dipertahankan dari operasi *opening*.



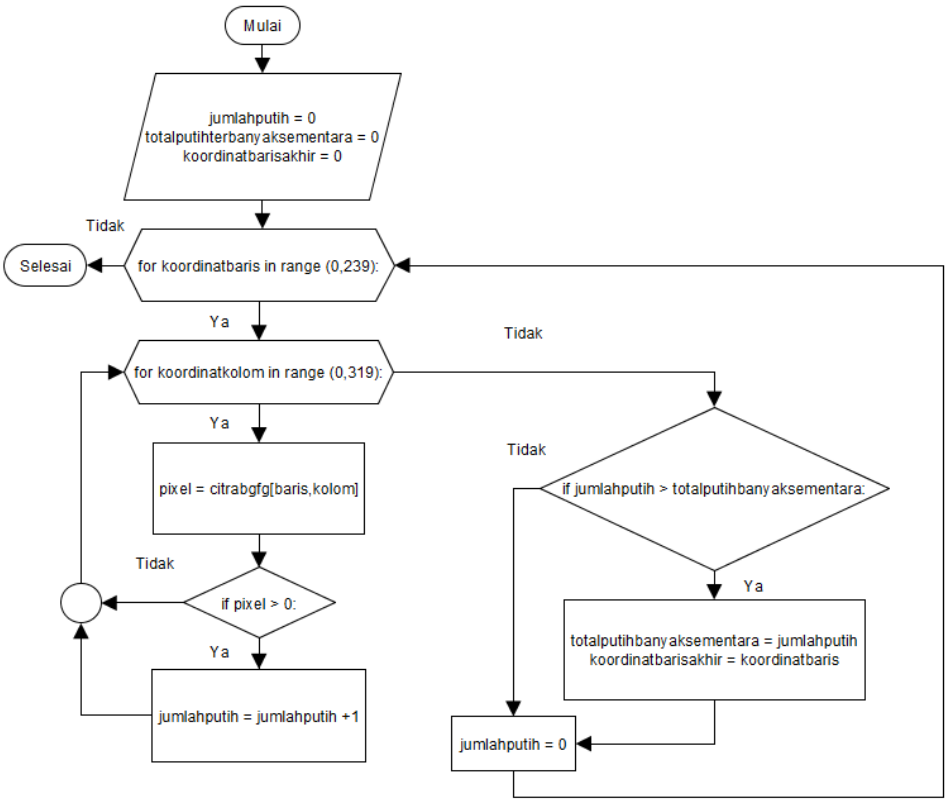
Gambar 5.10 Citra setelah operasi morfologi selesai dilakukan

Langkah terakhir yaitu melakukan proses penjumlahan pada citra *background* yang telah dimorfologi dengan citra *foreground*. Hal ini bertujuan untuk menampilkan garis tiap *slot* parkir dan kendaraan yang terdeteksi pada citra *foreground*.

Gambar 5.11 Citra hasil penjumlahan antara citra *background* dengan citra *foreground*

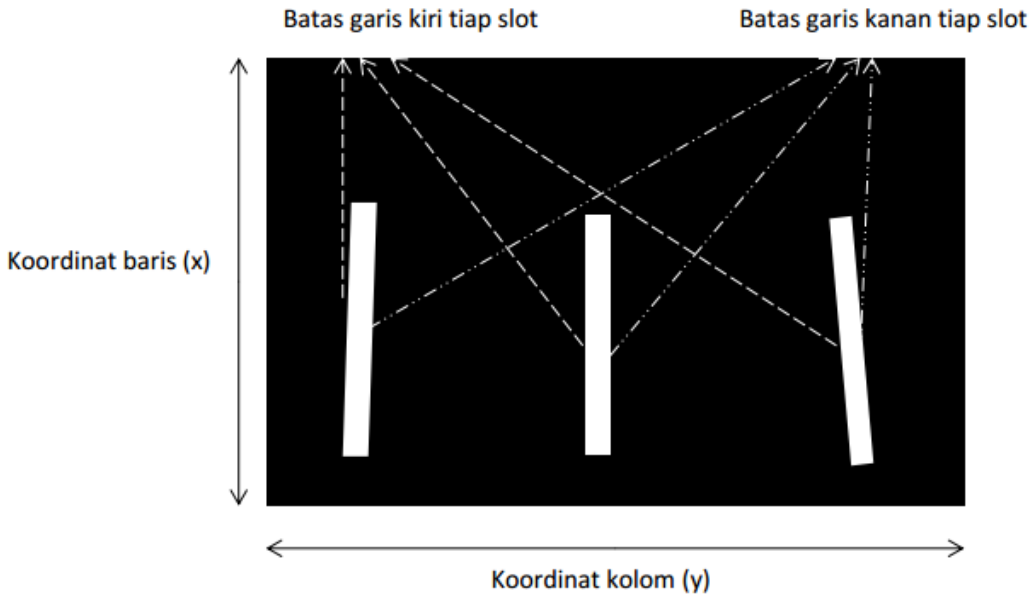
5.1.3.2 Perancangan fungsi mencari koordinat baris pada citra

Fungsi selanjutnya merupakan fungsi untuk mencari koordinat baris yang memiliki jumlah nilai *pixel* putih terbanyak pada citra. Hal ini dilakukan sebagai pertanda bahwa apabila pada suatu koordinat baris terdapat jumlah nilai *pixel* putih lebih banyak dari koordinat baris lainnya, maka di koordinat baris tersebut terdapat seluruh garis dari *slot* parkir yang akan dideteksi ketersediaan *slot*nya. Fungsi ini bekerja sesuai dengan diagram alir pada Gambar 5.12 berikut.



Gambar 5.12 Diagram alir fungsi mencari koordinat baris pada citra

Berikut ini merupakan skema penjelasan terkait koordinat baris, koordinat kolom, gariskiri dan gariskanan.



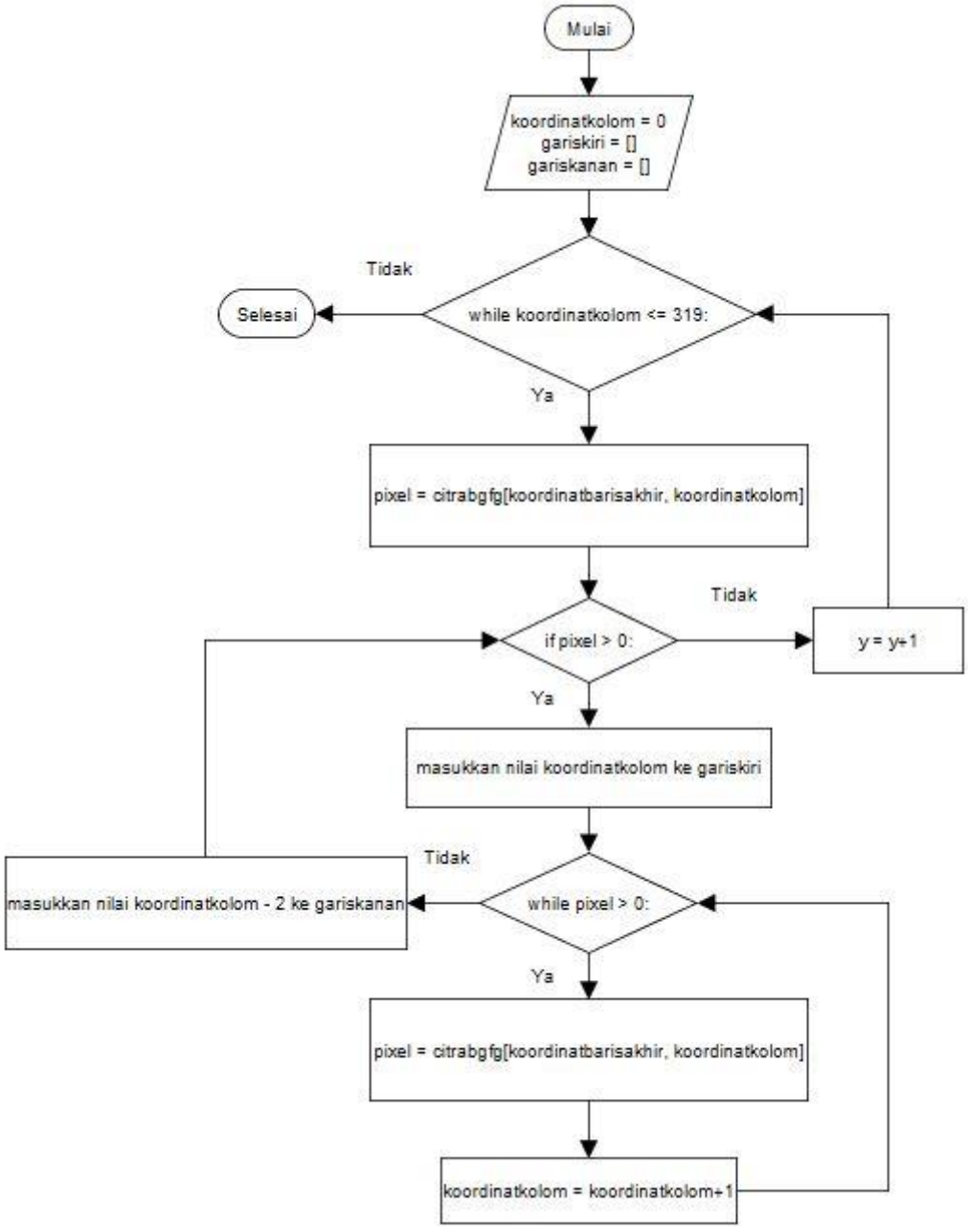
Gambar 5.13 Skema penjelasan

Setelah fungsi mencari koordinat baris dengan jumlah nilai *pixel* putih terbanyak selesai dijalankan, akan didapatkan titik koordinat baris dengan jumlah nilai *pixel* putih terbanyak yang nantinya akan menjadi acuan untuk fungsi selanjutnya.

Gambar 5.14 Hasil dari fungsi mencari koordinat baris pada citra

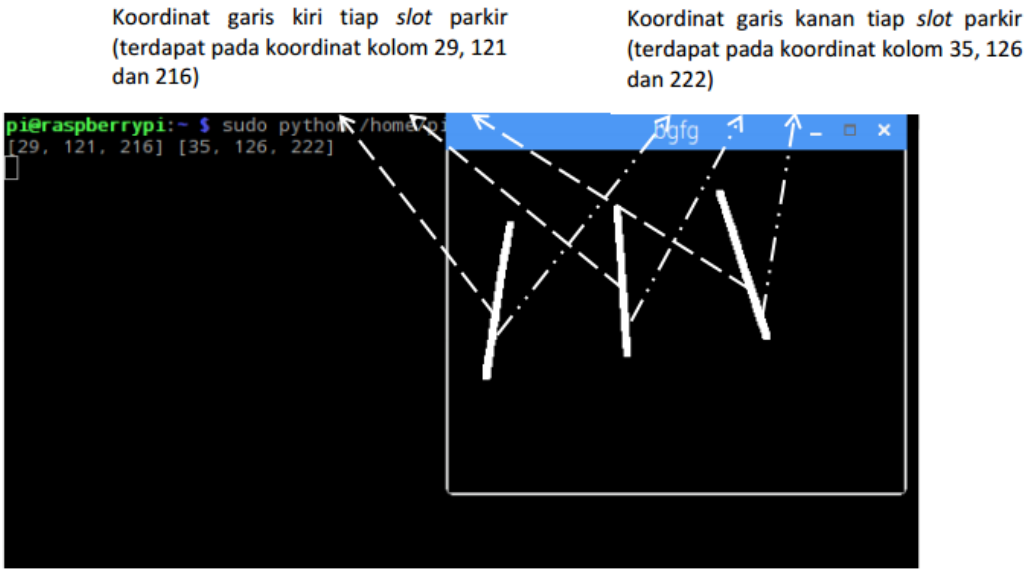
5.1.3.3 Perancangan fungsi mencari koordinat garis kiri dan garis kanan pada tiap *slot* parkir

Setelah didapatkan titik koordinat baris dengan jumlah nilai pixel putih terbanyak, fungsi selanjutnya merupakan fungsi untuk mencari koordinat garis kiri dan garis kanan dari tiap *slot* parkir. Fungsi ini berperan sebagai langkah awal dalam melakukan inisialisasi tiap *slot* parkir. Fungsi ini bekerja sesuai dengan diagram alir pada Gambar 5.15 berikut.



Gambar 5.15 Diagram alir fungsi mencari koordinat garis kiri dan garis kanan pada tiap *slot* parkir

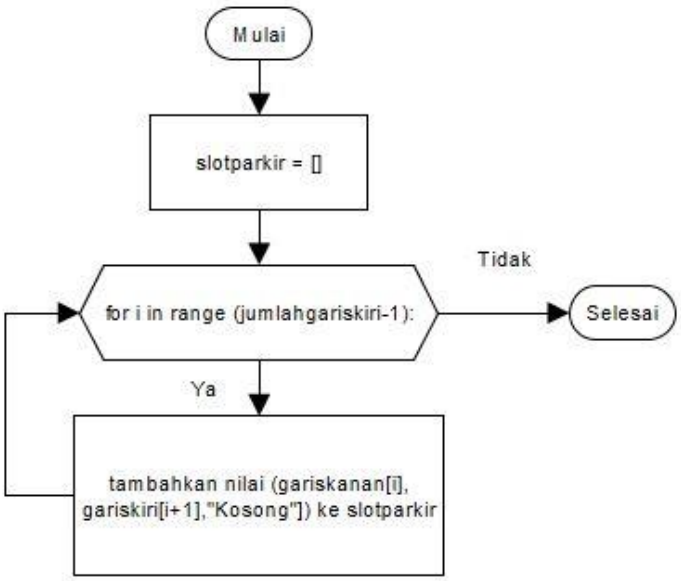
Berikut ini merupakan hasil setelah fungsi mencari garis kiri dan garis kanan di mana didapatkan koordinat setiap garis kiri dan garis kanan dari tiap *slot* parkir.



Gambar 5.16 Hasil dari fungsi mencari koordinat garis kiri dan garis kanan pada tiap *slot* parkir

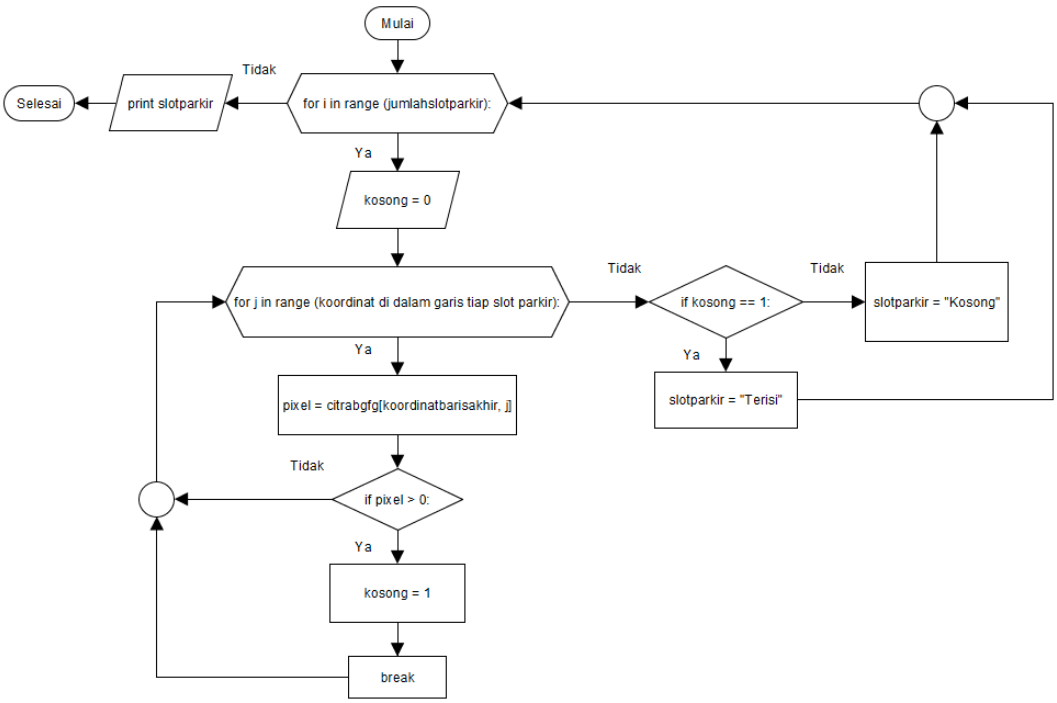
5.1.3.4 Perancangan fungsi penentuan ketersediaan *slot* parkir

Setelah didapatkan koordinat baris dan koordinat kolom untuk tiap *slot* parkir, langkah selanjutnya adalah melakukan inisialisasi *slot* parkir. Proses inisialisasi *slot* parkir bertujuan untuk mendapatkan nilai koordinat dari tiap *slot* parkir. Proses inisialisasi *slot* parkir dilakukan sesuai dengan diagram alir pada Gambar 5.17 berikut.



Gambar 5.17 Diagram alir proses inisialisasi nilai *slot* parkir

Setelah didapatkan nilai koordinat dari tiap *slot* parkir, dijalankan fungsi penentuan ketersedian *slot* parkir yang bertujuan untuk menentukan apakah *slot* parkir dalam keadaan kosong atau terisi. Fungsi tersebut bekerja sesuai dengan diagram alir pada Gambar 5.18 berikut.



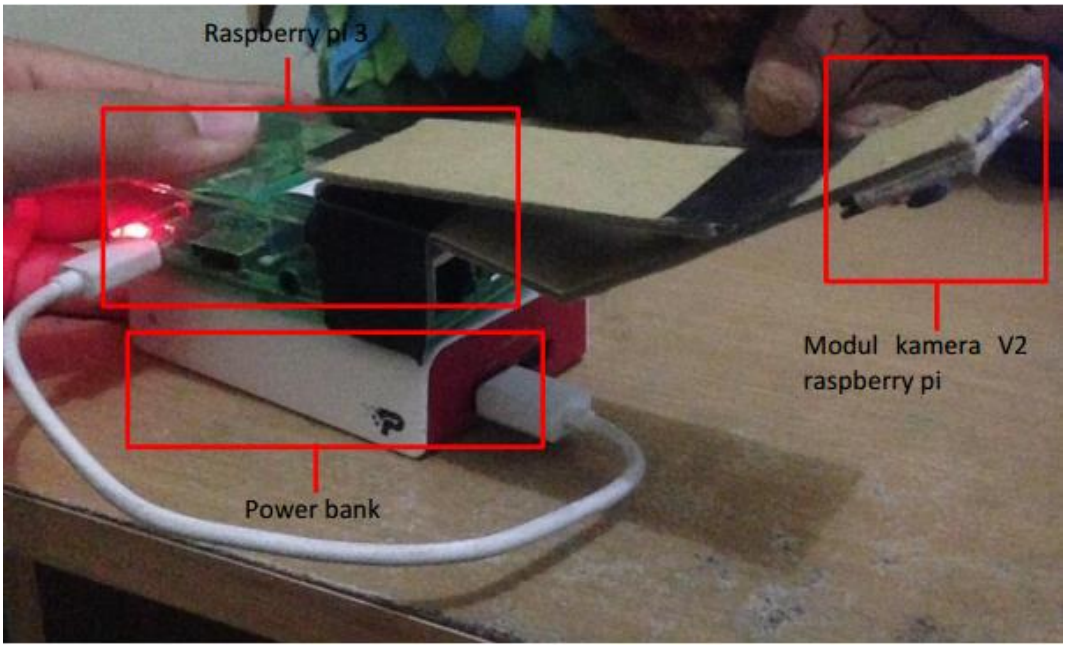
Gambar 5.18 Diagram alir fungsi penentuan ketersediaan *slot* parkir

5.2 Implementasi Sistem

Pada sub bab implementasi sistem akan dibahas mengenai langkah-langkah yang dilakukan dalam mengimplementasikan sistem berdasarkan perancangan yang telah dibuat pada sub bab sebelumnya.

5.2.1 Implementasi Perangkat Keras

Dalam melakukan tahap implementasi perangkat keras disesuaikan dengan perancangan perangkat keras yang telah dibuat pada sub bab 5.1.2, di mana digunakan sebuah modul kamera V2 raspberry pi yang terhubung dengan raspberry pi 3 melalui *port* kamera yang tersedia pada raspberry pi 3. Dikarenakan tidak adanya sumber tegangan untuk raspberry pi 3 di tempat sistem diletakkan, digunakanlah sebuah *power bank.* Berikut ini merupakan hasil dari implementasi perangkat keras yang ditunjukkan pada Gambar 5.19.



Gambar 5.19 Hasil implementasi perangkat keras

5.2.2 Implementasi Perangkat Lunak

Implementasi perangkat lunak merupakan tahap dalam membuat program untuk sistem berdasarkan perancangan yang ada pada sub bab 5.1.3. Program yang akan digunakan pada sistem diimplementasikan pada python IDLE menggunakan bahasa python. Di awal program dilakukan inisialisasi *library* numpy dan cv2, di mana *library* numpy digunakan untuk melakukan proses operasi aritmatika pada *array*. Sedangkan *library* cv2 merupakan *library* yang digunakan untuk memanggil fungsi-fungsi yang tersedia pada *library* openCV.

```
1  import numpy as np
2  import cv2
```

Gambar 5.20 Program inisialisasi *library*

5.2.2.1 Implementasi program inisialisasi citra

Pada tahap inisialisasi citra dilakukan beberapa tahapan proses, mulai dari pengambilan *input* citra, *resize* citra, konversi warna citra, proses *background subtraction*, proses *threshold*, proses morfologi dan penjumlahan citra. Proses-proses tersebut dilakukan pada python IDLE seperti pada Gambar 5.21 berikut.

```
1  cap = cv2.VideoCapture('/home/pi/sore1.mp4')
2  ret,frame = cap.read()
3  size = (320,240)
4  ukuranbaru = cv2.resize(frame, size, interpolation =
   cv2.INTER_AREA)
5  background = ukuranbaru
6  backgroundgray = cv2.cvtColor(background,
   cv2.COLOR_BGR2GRAY)
7  foregroundgray = cv2.cvtColor(ukuranbaru,
   cv2.COLOR_BGR2GRAY)
8  fgbg = cv2.subtract(foregroundgray, backgroundgray)
9  ret,thresh = cv2.threshold(fgbg,100,255,cv2.THRESH_BINARY)
10 ret,thresh1 =
   cv2.threshold(backgroundgray,180,255,cv2.THRESH_BINARY)
11 kernel = np.ones(3,3),np.uint8)
12 opening = cv2.morphologyEx(thresh1, cv2.MORPH_OPEN, kernel)
13 dilation = cv2.morphologyEx(opening, cv2.MORPH_DILATE,
   kernel)
14 bgfg = cv2.add(thresh, dilation)
```

Gambar 5.21 Program inisialisasi citra

Proses inisialisasi citra dimulai dengan mengambil *input* citra berupa video tempat parkir yang telah direkam sebelumnya dengan nama file "sore1.mp4" yang berada pada *folder* '/home/pi'. Kemudian *frame* pada video tersebut akan dibaca oleh sistem satu per satu. *Frame* yang telah dibaca akan di*resize* menjadi ukuran 320x240. *Frame* yang telah di*resize* kemudian dimasukkan kedalam variabel *ukuranbaru* untuk selanjutnya dimasukkan juga ke dalam variabel *background*. Selanjutnya *frame* yang telah di*resize* akan dikonversi warnanya dari citra RGB menjadi *grayscale*. Kemudian dilakukan proses *background subtraction* antara citra *foreground* dengan citra *background* yang telah dikonversi ke *grayscale*.

Citra hasil dari proses *background subtraction* selanjutnya akan diberikan *threshold* dengan persamaan sebagai berikut:

$$g(x,y) = \begin{cases} 255 \; if \; f(x,y) \geq 100 \\ 0 \; if \; f(x,y) < 100 \end{cases} \tag{5.1}$$

Dalam artian apabila nilai suatu *pixel* pada citra kurang dari nilai ambang yang diberikan (100), maka nilai *pixel*nya akan dirubah menjadi 0. Namun apabila nilai *pixel*nya lebih dari atau sama dengan 100, maka nilai *pixel*nya akan dirubah menjadi 255. Citra *background* juga diberikan nilai *threshold* dengan nilai ambang 180 dengan tujuan agar pada citra *background* hanya diambil garis-garis pada tiap *slot* parkir. Nilai *threshold* yang diberikan pada citra *background* berbeda dengan yang diberikan pada citra *foreground* (hasil dari proses *background subtraction*) dikarenakan pada citra *background*, garis-garis pada tiap *slot* parkir yang ingin dideteksi memiliki nilai *pixel* yang cenderung tinggi (mendekati 255), sedangkan pada citra *foreground,* yang ingin dideteksi adalah mobil yang akan menempati *slot* parkir. Nilai *threshold* yang diberikan pada citra *foreground* sebesar 100 dikarenakan apabila nilainya terlalu rendah, maka objek selain mobil akan ikut terdeteksi (semisal nilai *threshold* yang diberikan sebesar 25, maka objek selain mobil namun memiliki nilai *pixel* lebih dari atau sama dengan 25 akan terdeteksi sebagai mobil pada sistem), namun apabila nilainya

terlalu tinggi, maka mobil tidak akan terdeteksi pada sistem (objek yang terdeteksi hanya objek yang memiliki nilai *pixel* mendekati intensitas warna putih).

Langkah selanjutnya setelah citra selesai diberikan *threshold* adalah melakukan operasi morfologi berupa *opening* dan dilasi. Hal ini bertujuan untuk menghilangkan *noise* yang ada pada citra. Operasi morfologi dilakukan menggunakan *structure elements* berukuran 3x3 yang dideklarasikan dalam variabel *kernel*.

Langkah terakhir yaitu melakukan operasi aritmatika berupa penjumlahan pada citra *foreground* (citra hasil proses *background subtraction* yang telah diberikan *threshold*) dengan citra *background* (citra yang telah selesai dimorfologi). Hal ini bertujuan agar pada *output* dapat terlihat mobil yang memasuki *slot* parkir dan garis dari tiap *slot* parkir.

5.2.2.2 Implementasi program koordinat baris

Pada tahap implementasi program koordinat baris, sistem terlebih dahulu akan mencari koordinat baris yang memiliki jumlah nilai *pixel* putih terbanyak. Implementasi program dapat dilihat pada Gambar 5.22 berikut.

```
1   counter = 0 #penghitung pixel putih
2   z = 0 #total pixel putih
3   tengah = 0
4   for x in range (0,239):
5       for y in range (0,319):
6           px = bgfg[x,y]
7           if px > 0:
8               counter = counter+1
9       if counter > z:
10          z = counter
11          tengah = x
12      counter = 0
```

Gambar 5.22 Program mencari koordinat baris dengan jumlah nilai *pixel* putih terbanyak

Dapat dilihat pada baris 1-3 merupakan inisialisasi variabel *counter*, *z* dan *tengah* dimana ketiga variabel diberi nilai awal 0. Selanjutnya dilakukan perulangan untuk variabel *x* dengan *range* nilai variabel 0 hingga 239. Variabel *x* menandakan koordinat baris pada citra. Lalu dilakukan perulangan untuk variabel *y* dengan *range* nilai variabel 0 hingga 319. Variabel *y* menandakan koordinat kolom pada citra. Dalam perulangan *y* terdapat inisialisasi variabel *px* yang diisi dengan nilai koordinat (*x*,*y*) dari citra *bgfg*. Apabila nilai *pixel* pada koordinat (*x*,*y*) bernilai > 0, maka nilai dari variabel *counter* akan ditambahkan 1. Sistem akan terus melakukan perhitungan jumlah nilai *pixel* pada suatu baris mulai dari kolom 0 hingga kolom 319. Kemudian nilai pada variabel *counter* akan dibandingkan dengan nilai pada variabel *z*. Jika nilai pada variabel *counter* lebih besar dari nilai pada variabel *z*, maka nilai pada variabel *counter* akan dimasukkan ke dalam variabel *z*. Kemudian nilai koordinat *x* akan dimasukkan ke dalam variabel *tengah*. Kemudian nilai *counter* akan dirubah kembali menjadi 0 untuk selanjutnya sistem akan menghitung jumlah nilai *pixel* putih pada

koordinat baris selanjutnya hingga koordinat baris ke 239. Pada akhir dari proses ini akan didapatkan koordinat baris (x) dengan jumlah nilai *pixel* putih terbanyak yang titik koordinatnya disimpan pada variabel *tengah*.

5.2.2.3 Implementasi program koordinat garis kiri dan garis kanan tiap *slot* parkir

Setelah didapatkan nilai koordinat baris dengan jumlah nilai *pixel* putih terbanyak, selanjutnya sistem akan mencari koordinat garis kiri dan garis kanan tiap *slot* parkir yang diimplementasikan sesuai Gambar 5.23 berikut.

```
1  y = 0
2  gariskiri = []
3  gariskanan = []
4  while y <= 319:
5      px = bgfg[tengah,y]
6      if px > 0:
7          gariskiri.append(y)
8          while px > 0:
9              px = bgfg[tengah,y]
10             y = y+1
11         gariskanan.append(y-2)
12     y = y+1
```

Gambar 5.23 Program mencari koordinat garis kiri dan garis kanan tiap *slot* parkir

Pada baris 1-3 dilakukan inisialisasi awal variabel *y*, *gariskiri* dan *gariskanan* dimana variabel *y* menandakan koordinat kolom pada citra dan diberi nilai awal 0, sedangkan *gariskiri* dan *gariskanan* diinisialisasi sebagai *array*. Selanjutnya dilakukan perulangan selama nilai *y* <= 319. Dilakukan inisialisasi variabel *px* yang diisi dengan nilai koordinat (*tengah*,*y*) pada citra *bgfg* dimana nilai koordinat baris (nilai pada variabel *tengah*) didapatkan dari proses sebelumnya. Kemudian program akan membandingkan nilai *pixel* pada tiap koordinat kolom (*y*). Jika nilai *pixel*nya > 0, maka koordinat tersebut akan ditambahkan ke dalam variabel *gariskiri*. Proses berikutnya, program akan terus membandingkan nilai *pixel* pada koordinat kolom selanjutnya. Jika nilai *pixel* pada koordinat kolom selanjutnya masih bernilai > 0, maka nilai koordinat kolom akan ditambah 1 (digeser ke koordinat kolom selanjutnya) hingga nilai *pixel*nya 0. Selanjutnya program akan menambahkan nilai koordinat kolom terakhir - 2 ke dalam variabel *gariskanan*. Setelah itu program kembali akan menggeser nilai koordinat kolom hingga koordinat 319. Pada akhir dari program ini akan didapatkan nilai koordinat garis kiri dan garis kanan dari tiap garis yang ada pada *slot* parkir.

5.2.2.4 Implementasi program inisialisasi nilai *slot* parkir

Setelah didapatkan koordinat baris dengan jumlah nilai *pixel* putih terbanyak dan koordinat garis kiri dan garis kanan tiap garis *slot* parkir, yang dilakukan selanjutnya adalah melakukan inisialisasi *slot* parkir sesuai dengan Gambar 5.24 berikut.

```
1  slotparkir = []
2  for i in range (len(gariskiri)-1):

3  slotparkir.append([gariskanan[i],gariskiri[i+1],"Kosong"])
```

Gambar 5.24 Program inisialisasi nilai *slot* parkir

Pada awal program dilakukan inisialisasi awal variabel *slotparkir* yang didefinisikan sebagai *array*. Kemudian dilakukan perulangan untuk variabel *i* dengan *range* jumlah nilai yang ada pada variabel *gariskiri* dikurangi 1. Selanjutnya nilai pada variabel *gariskanan[i]* dan *gariskiri[i+1]* serta kondisi awal *slot* parkir, yaitu "Kosong" akan dimasukkan ke dalam variabel *slotparkir*. Sehingga *slotparkir[0]* akan berisi *gariskanan[0]*, *gariskiri[0+1]*, dan "Kosong".

5.2.2.5 Implementasi program penentuan ketersediaan *slot* parkir

Langkah selanjutnya yaitu menentukan ketersediaan *slot* parkir yang diimplementasikan sesuai dengan program yang ada pada Gambar 5.25 berikut.

```
1  for i in range (len(slotparkir)):
2      kosong = 0
3      for j in range (slotparkir[i][0]+1,slotparkir[i][1]-1):
4          px = bgfg[tengah,j]
5          if px > 0:
6             kosong = 1
7             break
8      if kosong == 1:
9           slotparkir[i][2] = "Terisi"
10     else:
11          slotparkir[i][2] = "Kosong"
12 print slotparkir
```

Gambar 5.25 Program penentuan ketersediaan *slot* parkir

Dilakukan perulangan untuk inisialisasi variabel *i* dengan nilai *i* merupakan *range* dari jumlah index yang ada pada variabel *slotparkir*. Kemudian pada perulangan *i* akan dilakukan inisialisasi nilai awal variabel *kosong* dengan nilai 0 dan perulangan untuk inisialisasi variabel *j* dengan nilai variabel *slotparkir[i][0]*+1 hingga nilai variabel *slotparkir[i][1]*-1, dimana semisal nilai pada *slotparkir[i][0]* bernilai 40 dan nilai pada *slotparkir[i][1]* bernilai 90 maka variabel *j* akan berisi nilai 41 hingga 89. Kemudian dilakukan inisialisasi nilai untuk variabel *px* yang berisi koordinat(*tengah,j*). Selanjutnya program akan membandingkan nilai *pixel*, di mana jika nilai *pixel* lebih besar dari 0 maka nilai pada variabel *kosong* akan dirubah menjadi 1 dan program akan melakukan *break*. Kemudian program akan membandingkan nilai variabel *kosong*. Jika variabel *kosong* bernilai 1, maka *slot* parkir dalam kondisi terisi, namun apabila variabel *kosong* bernilai 0, maka *slot* parkir dalam kondisi kosong.