BAB III
ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisis Sistem

Analisis sistem merupakan penguraian dari suatu sistem informasi yang utuh kedalam bagian – bagian komponennya dengan maksud untuk mengidentifikasi dan mengevaluasi permasalahan, kebutuhan serta hambatan yang terjadi. Menentukan kebutuhan yang sesuai dengan kebutuhan – kebutuhan yang diharapkan.

3.1.1 Analisis Masalah

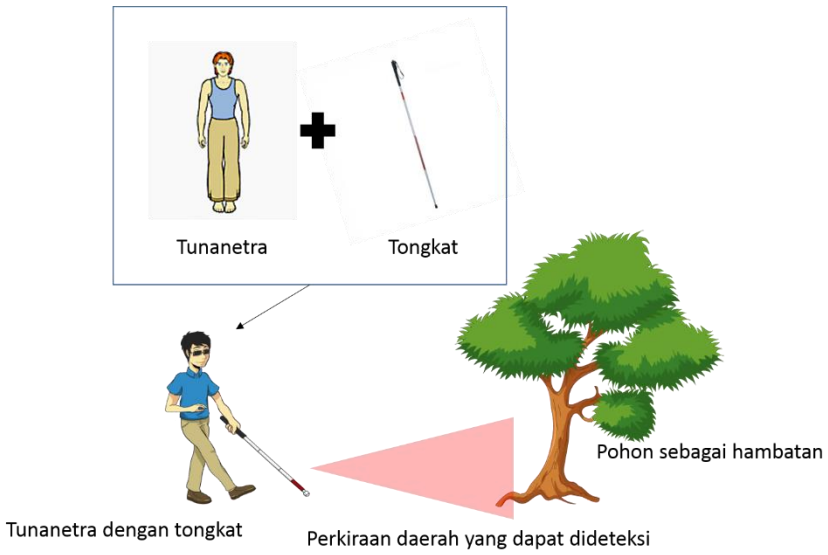
Permasalahan pada penelitian ini adalah melengkapi kekurangan yang dihadapi seorang tunanetra ketika berjalan menggunakan tongkat yang tidak dapat mengakomodasi seluruh kebutuhan. Hal ini dikarenakan tongkat hanya dapat mendeteksi hambatan yang berada di bagian bawah pinggang penderita tunanetra. Dengan adanya masalah diatas penelitian ini bertujuan untuk memanfaatkan *smartphone* Android agar dapat digunakan untuk melengkapi keberadaan tongkat sebagai pendeteksi hambatan berjalan bagi tunanetra.

3.1.2 Analisis Prosedur yang Berjalan

Berdasarkan hasil wawancara langsung didapatkan prosedur yang dilakukan seorang tunanetra ketika menghadapi situasi jalan atau rute yang baru dilewati. Langkah yang dilakukan tunanetra sebelum dan ketika berjalan ialah:

1. Menanyakan situasi medan jalan yang akan dilewati kepada orang normal.
2. Menggunakan alat bantu tongkat sebagai pendeteksi hambatan.

Prosedur penggunaan alat bantu tongkat yang digunakan saat ini sebagai pendeteksi hambatan dapat dilihat pada gambar berikut:



Gambar 3.1 Alur Prosedur yang Berjalan

Dari gambaran tersebut, terlihat bahwa alat yang digunakan tunanetra saat ini yaitu tongkat masih terdapat kekurangan ketika akan mendeteksi hambatan. Hal itu berdasarkan wawancara bersama Pak Halim seorang penyandang tunanetra yang menceritakan daerah mana yang dapat dideteksi oleh tongkat.

Maka diperlukan alat bantu lain sebagai pendamping alat bantu saat ini yaitu tongkat agar penyandang tunanetra dapat mendeteksi hambatan yang tidak terdeteksi oleh tongkat. Alat tersebut memiliki karakteristik sebagai berikut:

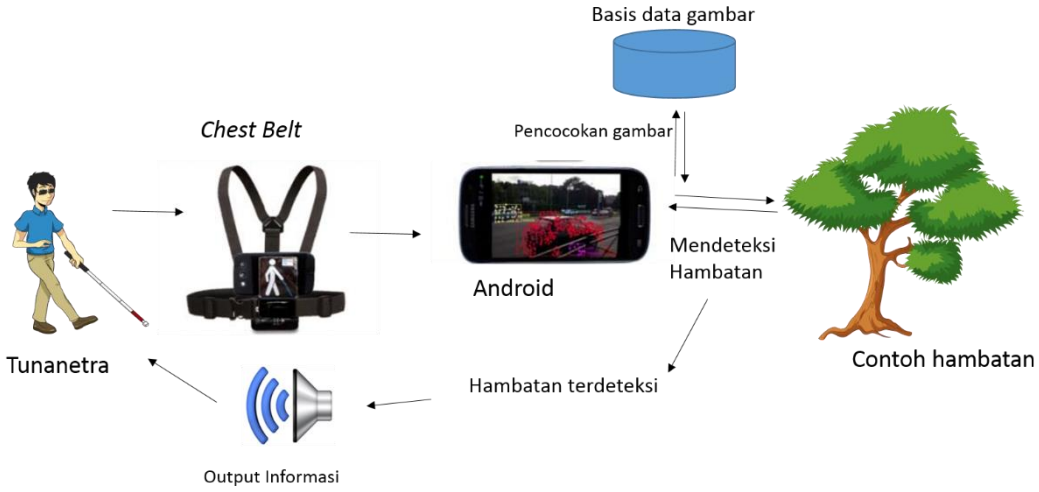
1. Dapat mendeteksi wilayah yang tidak dapat dijangkau oleh tongkat.
2. Memanfaatkan teknologi yang telah ada.
3. Dapat digunakan dengan baik oleh tunanetra.

Berdasarkan karakteristik tersebut maka pendekatan teknologi yang tepat terdapat pada pemanfaatan kamera pada smartphone Android untuk mendeteksi hambatan di wilayah yang tidak terjangkau oleh tongkat. Pembangunan aplikasi pada Android untuk memanfaatkan kamera menggunakan *library* OpenCV.

3.1.3 Analisis Arsitektur Sistem

Pada arsitektur sistem yang dibangun terdiri dari beberapa objek diantaranya: Tunanetra sebagai pengguna, *Chest Belt* sebagai alat yang digunakan

untuk memasang perangkat android pada badan, serta perangkat Android yang digunakan untuk mendeteksi hambatan. Gambaran arsitektur sistem dapat dilihat pada gambar berikut:



Gambar 3.2 Arsitektur Sistem yang Dibangun

Sistem yang dibangun adalah suatu aplikasi berbasis *smartphone* Android dengan memanfaatkan kamera sebagai pendeteksi hambatan menggunakan metode ORB pada *library* OpenCV. Metode ORB berguna untuk mendeteksi dan mendeksripsi fitur pada hambatan yang akan dideteksi lalu dilakukan pencocokan fitur pada gambar. Jika ditemukan hambatan yang cocok, maka sistem akan memberikan informasi kepada tunanetra.

3.1.4 Analisis Kebutuhan *Tools/Library* OpenCV

Pada saat pembangunan perangkat lunak "Deteksi Hambatan Berjalan" diperlukan perangkat lunak seperti *library* serta *tools* khusus yang dapat digunakan untuk mendeteksi hambatan dan mengenali hambatan. *Library* yang digunakan untuk pembangunan aplikasi ini ialah OpenCV. Berikut ini merupakan perbandingan *library* yang dapat digunakan untuk pembangunan aplikasi "Deteksi Hambatan Berjalan":

Tabel 3.1 Perbandingan Library

No.	Parameter	Tools/Library		
		OpenCV	BoofCV	CVIPTools
1.	Lisensi	BSD License	Apache 2.0	Tidak diketahui
2.	Dukungan Bahasa	Python, C/C++, Java	Java	C/C++, C#
3.	Modul Deteksi Fitur	ORB, FAST, BRISK, AGAST	Fast Hessian	Tidak Diketahui
4.	Dukungan Platform	Windows, Linux, Android, iOS, MacOS	Android	Windows, Linux

Kelebihan OpenCV sebagai *library* yang tepat digunakan untuk pembangunan aplikasi ini ialah:

1. Lisensi gratis dengan dukungan di platform yang lebih banyak.
2. Dukungan pengembang dan pengguna di forum OpenCV maupun di forum interaksi lain sangatlah aktif.
3. Metode yang dapat digunakan untuk mendeteksi fitur terdapat berbagai macam.

Selain kelebihan diatas, OpenCV memiliki kekurangan diantaranya diharuskan melakukan *pre-install* OpenCV Manager pada perangkat Android yang akan digunakan. Sebenarnya hal ini dapat diganti dengan instalasi *library* statis, namun diperlukan modul tambahan lain yang tidak dijelaskan dalam penelitian ini.

3.1.5 Analisis Metode ORB Untuk Mendeteksi Hambatan

Metode ORB merupakan metode yang digunakan untuk mendeteksi hambatan dalam pembangunan aplikasi ini. ORB memiliki 2 fungsi utama yaitu deteksi fitur dan deskripsi fitur. Pada deteksi fitur ORB mengadopsi metode FAST-9 yang telah dimodifikasi agar dapat mendeteksi fitur pada citra sebaik mungkin.

Berikut merupakan langkah – langkah yang dilakukan metode ORB untuk mendeteksi fitur yang terdapat pada citra:

1. Melakukan *segment test algorithm* di titik p pada koordinat (Xp, Yp) dalam citra dengan menggunakan lingkaran Bresenham 16 piksel.
2. Menerapkan *decision tree* pada setiap titik piksel tetangga dengan atribut:

$$1, jika\ intensitas\ piksel\ tetangga\ >\ (p\ +\ threshold)$$
$$-1, jika\ intensitas\ piksel\ tetangga\ <\ (p\ -\ threshold)$$
$$0, jika\ intensitas\ piksel\ tetangga\ =\ titik\ p.$$

3. Jika lebih dari 9 (karena FAST-9) piksel tetangga terhadap titik pusat p yang lebih cerah atau gelap maka titik p disebut sebuah fitur.
4. Lakukan klasifikasi dan ekstrak atribut untuk semua piksel pada citra.
5. Untuk setiap piksel, pilih atribut seperti pada tahap 2 menjadi sebuah sublist.
6. Pada setiap titik yang dijadikan sublist, lakukan sublist lagi ke dalam 3 bagian yang lebih mendalam.
7. Ulangi langkah 3 hingga sublist berisi hanya bagian titik piksel yang dinyatakan sebagai sebuah fitur.

Pada deskripsi atau ekstraksi fitur, ORB mengadopsi metode BRIEF yang didesain ulang agar dapat mendeskripsi fitur pada citra. Berikut merupakan langkah – langkah yang dilakukan metode ORB untuk mendeskripsi fitur pada citra:

8. Melakukan *binary test* pada $\tau\ (p: x, y)$, lalu memberi nilai dengan ketentuan:

$$1, jika\ p(x)\ <\ p(y)$$
$$0, untuk\ kemungkinan\ lain.$$

   Dimana $p(x)$ merupakan intensitas piksel yang dilakukan *smoothing* dari p pada titik x.
9. Lalu terapkan rumus BRIEF pada dimensi nd:

$$f_{nd}(p) := \sum_{1 \le i \le nd} 2^{i-1}\ \tau(p: x_i, y_i)$$

10. Untuk setiap $S * S$, piksel di *smoothing* dengan Gaussian dan pilih piksel menggunakan *binary test*.

11. Melakukan *spatial arrangement of the binary test* untuk setiap titik piksel.

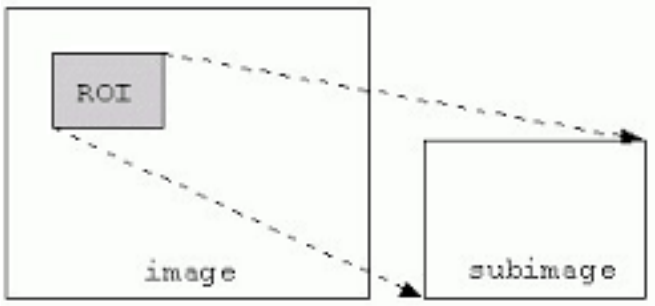
3.1.6 Analisis Sudut Pandang Kamera

Pembangunan aplikasi pendeteksi hambatan ini memanfaatkan komponen kamera yang berada pada perangkat Android. Kamera pada Android sendiri memiliki sudut pandang yang cukup luas dengan ukuran bervariasi tergantung tipe kamera. Standar ukuran yang digunakan dalam menghitung sudut kamera ialah derajat.

Semakin luas ukuran sudut kamera maka semakin luas objek yang dapat ditangkap oleh kamera. Hal ini sangat berpengaruh terhadap pendeteksian objek yang dilakukan dalam penelitian ini. Objek yang berada di luar jangkauan tunanetra pun memiliki kemungkinan tertangkap oleh kamera.

Berdasarkan masalah tersebut maka diperlukan suatu metode atau langkah khusus yang diterapkan dalam membangun aplikasi pendeteksi objek ini agar dapat membatasi sudut pandang terhadap objek yang ditangkap kamera. Hal ini dapat diselesaikan dengan memberi area khusus terhadap sudut pandang kamera. Pemberian area ini dapat menggunakan metode ROI (*Region of Interest*). Dengan menggunakan metode ROI ini luas sudut pandang kamera dapat dibatasi sesuai kebutuhan.

ROI (*Region of Interest*) merupakan sebuah area khusus yang didefinisikan atas tujuan tertentu. Area khusus ini dapat menjadi sebuah daerah yang berisi informasi – informasi penting. Dalam hal mendeteksi objek pada kamera, ROI ini sangat berguna untuk membatasi wilayah citra dalam citra input agar ruang komputasi menjadi lebih efektif.



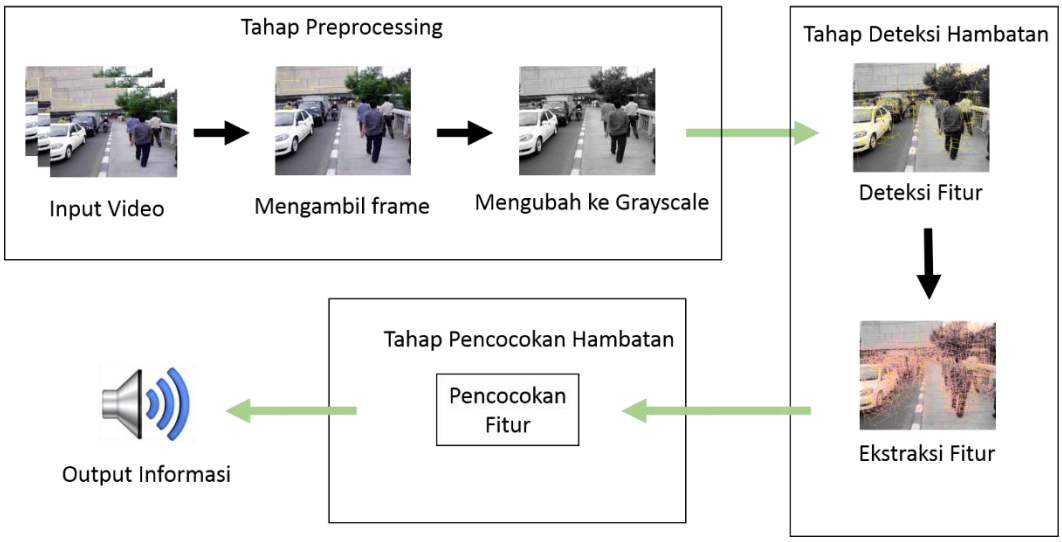
Gambar 3.3 ROI (*Region of Interest*)

Sumber gambar: xuebuyuan.com

3.1.7 Analisis Aplikasi yang Akan Dibangun

Aplikasi yang akan dibangun adalah sebuah aplikasi berbasis Android yang dapat mendeteksi hambatan yang ditemui tunanetra ketika berjalan. Aplikasi ini berguna untuk melengkapi alat yang digunakan tunanetra saat ini yaitu tongkat. Proses yang digunakan dalam aplikasi ini ialah proses deteksi dan pencocokan hambatan. Hal ini dapat dilakukan dengan melakukan pengolahan citra digital terhadap jalan yang akan dilalui oleh tunanetra. Setelah pengolahan citra digital, aplikasi akan memberikan informasi berupa suara kepada tunanetra.

Untuk lebih jelasnya proses yang akan terjadi dalam aplikasi yang akan dibangun dijelaskan dalam gambar 3.3 berikut ini:



Gambar 3.4 Proses Deteksi Hambatan Berjalan

Dalam setiap tahapan yang dijelaskan pada gambar 3.3, diimplementasikan menggunakan *library* dari OpenCV.

3.1.7.1 Tahap Preprocessing

Tahap preprocessing merupakan tahap persiapan untuk proses awal. Input yang berupa video diproses per frame. Preprocessing meliputi 3 tahap, yaitu input video, mengambil frame dari video dan mengubah citra RGB ke grayscale.

3.1.7.1.1 Input Video

Langkah pertama pada setiap pengolahan citra ialah mengakuisisi citra dari kamera. Kamera disini mengacu kepada kamera yang terdapat pada *smartphone*. Kamera mengubah analog cahaya kedalam sebuah array intensitas data piksel.

Dalam tahap ini *library* OpenCV yang digunakan ialah *class CameraBridgeViewBase*. *CameraBridgeViewBase* merupakan sebuah *class* yang berinteraksi antara kamera pada android dan *library* OpenCV. *Class* ini berguna untuk menampilkan video *preview* dalam tampilan antarmuka android.

Berikut merupakan contoh pengambilan video *preview* pada antarmuka android:



Gambar 3.5 Contoh Video *Preview* pada Android

Contoh penggunaan *class CameraBridgeViewBase* dalam pembangunan aplikasi ini dapat dilihat sebagai berikut:

Baris	Kode Program
1	`private CameraBridgeViewBase mOpenCvCameraView;`
2	
3	`mOpenCvCameraView.enableView();`
4	
5	`mOpenCvCameraView = (CameraBridgeViewBase)`
6	`findViewById(R.id.DeteksiHambatanView);`
7	`mOpenCvCameraView.setVisibility(SurfaceView.VISIBLE);`
8	`mOpenCvCameraView.setCvCameraViewListener(this);`
9	
10	`mOpenCvCameraView.disableView();`

Penjelasan kode program diatas ialah:

Pada baris 1 terdapat `mOpenCvCameraView`, merupakan definisi dari *class CameraBridgeViewBase*.

Untuk mengaktifkan *CameraBridgeViewBase* digunakan *method* bawaan yaitu *enableView().*

Pada baris 5 terdapat sebuah definisi layout dengan id ***DeteksiHambatanView*** yang dilakukan *casting* terhadapat *CameraBridgeViewBase.*

Lalu pada baris 7 terdapat *method* bawaan *setVisibilty()* yang diturunkan dari *class SurfaceView* milik Android. *Method* ini berguna untuk melakukan rendering *camera* terhadap layar.

Pada baris 8 terdapat *method* bawaan dari *class CameraBridgeViewBase* untuk mengaktifkan listener yang terdapat dalam *context* yang digunakan. Biasanya *method* ini menampilkan hasil pengolahan citra yang dilakukan OpenCV.

Untuk menonaktifkan *class CameraBridgeViewBase* digunakan *method* bawaan yaitu *disableView().*

Penggunan kode program yang lebih jelas dan terperinci dapat dilihat pada lampiran KameraActivity.Java.

3.1.7.1.2Mengambil Frame dari Video

Video merupakan gabungan gambar – gambar mati yang dibaca berurutan dalam satu waktu dengan kecepatan tertentu. Gambar – gambar yang digabung tersebut dinamakan frame dan kecepatan pembacaan gambar disebut dengan frame rate, dengan satuan fps (frame per second). Karena dimainkan dalam kecepatan yang tinggi maka tercipta ilusi gerak yang halus, semakin besar nilai frame rate maka akan semakin halus pergerakan yang ditampilkan.

Untuk dapat mengambil sebuah *frame* yang ditangkap oleh kamera dan ditampilkan video *preview* yang telah dijelaskan pada tahap 3.1.3.1.1, digunakan sebuah *abstract function* pada *library* OpenCV yaitu *OnCameraFrame* yang merupakan bagian dari *interface CVCameraViewListener2*.

Contoh penggunaan *abstract function OnCameraFrame* yang digunakan dalam aplikasi ini ialah sebagai berikut:

Baris	Kode Program
1	`public class KameraActivity extends Activity implements`
2	`CvCameraViewListener2`
3	
4	`public Mat onCameraFrame(CvCameraViewFrame inputFrame) {`
5	
6	`mRgba = inputFrame.rgba();`
7	
8	`return mRgba;`
9	`}`

Penjelasan kode program diatas ialah:

Pada baris 1 terdapat sebuah *class KameraActivity* yang mengambil fungsi dari *class Activity* bawaan Android. *Class* ini mengimplementasikan *interface CvCameraViewListener2* agar dapat menggunakan *method* dan *function* yang tersedia.

Pada baris 4 terdapat sebuah *abstract function OnCameraFrame* yang bertipe *Mat* untuk mengambil *frame* dari *CvCameraViewFrame* yang merupakan tipe dari *class CameraBridgeViewBase*.

Pada baris 6 terdapat variabel `mRgba` yang diinisialisasi method *rgba()* dari tipe *CvCameraViewFrame* pada *class CameraBridgeViewBase*.

Untuk menampilkan hasil listener kepada *camera preview* maka perlu memberikan nilai *return* yang bertipe *Mat* yang terlihat pada baris 8 yaitu `mRgba`.

Penggunan kode program yang lebih jelas dan terperinci dapat dilihat pada lampiran KameraActivity.Java

3.1.7.1.3Mengubah Citra RGB ke Grayscale

Grayscale adalah proses pengubahan sebuah gambar menjadi gambar keabuan. Dalam tiap *pixel* pada gambar mengandung satu *sample* atau sinyal, yang hanya membawa informasi intensitas.

Berikut merupakan salah satu contoh citra RGB yang diambil sebagai citra uji:



Gambar 3.6 Citra Uji

Lalu citra uji diatas dirubah ke grayscale dan diperoleh hasil citra seperti pada gambar 3.6:



Gambar 3.7 Citra Uji Grayscale

Untuk mendapatkan citra grayscale digunakan sebuah implementasi *method* yang terdapat pada *library* OpenCV. *Method* yang digunakan ialah *inputFrame.gray()* yang diimplementasikan didalam sebuah prosedur *OnCameraFrame.*

Contoh penggunaan *method inputFrame.gray()* yang digunakan dalam pembangunan aplikasi ini ialah:

Baris	Kode Program
1	`public Mat onCameraFrame(CvCameraViewFrame inputFrame) {`
2	
3	`mGray = inputFrame.gray();`
4	
5	`return mGray;`
6	`}`

Penjelasan kode program diatas ialah:

Pada baris 1 terdapat sebuah *abstract function OnCameraFrame* yang bertipe *Mat* untuk mengambil *frame* dari *CvCameraViewFrame* yang merupakan tipe dari *class CameraBridgeViewBase*.

Pada baris 3 terdapat variabel `mGray` yang diinisialisasi method *gray()* dari tipe *CvCameraViewFrame* pada *class CameraBridgeViewBase.*

Untuk menampilkan hasil listener kepada *camera preview* maka perlu memberikan nilai *return* yang bertipe *Mat* yang terlihat pada baris 5 yaitu `mGray`.

Penggunan kode program yang lebih jelas dan terperinci dapat dilihat pada lampiran KameraActivity.Java

3.1.7.2 Tahap Deteksi Hambatan

Setelah mendapatkan citra grayscale yang siap diolah, maka tahap selanjutnya ialah tahap deteksi hambatan. Pada tahap ini diimplementasikan metode atau algoritma yang dapat mendeteksi fitur dan mengekstraksi fitur yang terdapat pada *library* OpenCV.

3.1.7.2.1Deteksi Tepi

Deteksi Tepi merupakan kegiatan untuk menemukan tepi atau garis pada suatu citra yang akan diolah. Tepi dan garis – garis tersebut akan membentuk objek gambar dan akan memperjelas bentuk objek pada bagian tersebut. Tujuan dari deteksi tepi ini ialah mengidentifikasi objek agar dapat mudah dikenali dan disederhanakan bentuknya. Metode yang digunakan untuk deteksi tepi ini ialah *Canny Edge Detection*. Metode *Canny Edge Detection* menggunakan algoritma multi-tahap untuk mendeteksi tepi pada sebuah citra. Pada Gambar dibawah ini merupakan contoh dari implementasi metode *Canny Edge Detection*:



Sumber gambar: mathworks.com

Gambar 3.8 Canny Edge Detection

Implementasi metode *Canny* pada pembangunan aplikasi ini terdapat pada *library* OpenCV dengan menggunakan *class Imgproc*. *Class Imgproc* merupakan *class* yang dapat digunakan untuk mengolah dan melakukan komputasi pada citra.

Contoh penggunaan *method Canny* pada *class Imgproc* yang digunakan untuk membangun aplikasi ini ialah:

Baris	Kode Program
1	`mIntermediateMat = new Mat(height, width, CvType.CV_8UC4);`
2	
3	`public Mat onCameraFrame(CvCameraViewFrame inputFrame) {`
4	
5	`mRgba = inputFrame.rgba();`
6	`Imgproc.Canny(inputFrame.gray(), mIntermediateMat,`
7	`80, 100);`
8	`Imgproc.cvtColor(mIntermediateMat, mRgba,`
9	`Imgproc.COLOR_GRAY2RGBA, 4);`
10	
11	`return mRgba;`
	`}`

Penjelasan kode program diatas ialah:

Seperti penjelasan yang terdapat pada tahap 3.1.4.1.3, setelah frame diambil maka dilakukan pengolahan pada citra seperti pada baris 5. *Imgproc* merupakan sebuah *class library* yang terdapat dalam OpenCV dan memiliki *method Canny()* yang memiliki 4 parameter diantaranya `inputFrame.gray()` merupakan sebuah tipe Mat bernilai *grayscale* sebagai citra input untuk *method Canny*, lalu parameter berikutnya `mIntermediateMat` merupakan sebuah variabel bertipe Mat yang telah didefinisikan sebelumnya dengan `new Mat(height, width, CvType.CV_8UC4);` pad baris 1 sebagai input citra pembanding. `CvType.CV_8UC4` memiliki maksud mendefinisikan kedalaman warna 8 bit, jumlah *channel* 4 dan *layout channel unsigned char* yang akan digunakan Parameter ketiga dan keempat merupakan sebuah nilai yang diinginkan sebagai nilai *thresholding maximum* dan *minimum*.

Pada baris 8 terdapat *method cvtColor()* yang merupakan bagian dari *class Imgproc* yang bertujuan untuk mengubah suatu *color space* pada citra.

Penggunan kode program yang lebih jelas dan terperinci dapat dilihat pada lampiran KameraActivity.Java

3.1.7.2.2Deteksi Fitur

Setelah mendapatkan pola atau bentuk objek maka langkah selanjutnya ialah mengenali objek dengan cara deteksi fitur. Deteksi Fitur merupakan suatu kegiatan untuk menemukan fitur, dalam hal ini *interest points*, yang terdapat dalam citra. Fitur yang dideteksi ialah *corner, edge,* dan lainnya. Kriteria yang penting dalam mendeteksi fitur ialah dapat diulang dan invarian. Maksudnya tahap ini mengharuskan ditemukannya fitur yang sama dalam citra yang berbeda. *Output* dalam tahap ini ialah deskripsi numerik pada suatu wilayah.

Pada *library* OpenCV untuk dapat mendeteksi fitur terdapat beberapa metode yang dapat dilakukan. Salah satunya menggunakan algoritma ORB Feature Detector. Pada implementasinya di OpenCV algoritma ini tersimpan dalam bahasa C/C++, sedangkan untuk dapat menggunakannya di android bahasa yang tersimpan seharusnya dalam bahasa Java. Hal ini dapat dimungkinkan dengan menggunakan Android NDK. Android NDK sendiri merupakan sebuah *tools* yang dapat digunakan untuk memungkinkan pengembang perangkat lunak mengimplementasikan kode native seperti C dan C++.

Setelah algoritma diatas tersimpan dalam *library*, maka langkah selanjutnya ialah memanggil prosedur dengan cara native pada Android. Pada langkah inilah Android NDK diperlukan untuk mengkompilasi bahasa C/C++ ke dalam bahasa Java. Untuk memperlihatkan nilai numerik fitur yang terdapat pada sebuah frame, dibuatlah lingkaran – lingkaran kecil sebagai penanda.

Pada tahap deteksi fitur, metode yang digunakan ialah ORB. ORB merupakan metode yang dapat mendeteksi fitur pada citra dengan menggunakan basis metode FAST dengan modifikasi tambahan.

Implementasi metode ORB ini diterapkan dalam suatu file berekstensi .cpp yang dikenali melalui Android NDK. Modul pada OpenCV yang digunakan untuk menerapkan metode ini ialah *features2d*. Modul ini berisikan algoritma dan metode yang diperlukan untuk dapat mendeteksi fitur.

Contoh penggunaan metode ORB untuk mendeteksi fitur yang digunakan dalam aplikasi ini ialah:

Baris	Kode Program
1	`public Mat onCameraFrame(CvCameraViewFrame inputFrame) {`
2	
3	`mRgba = inputFrame.rgba();`
4	`mGray = inputFrame.gray();`
5	`FindFeatures(mGray.getNativeObjAddr(),mRgba.getNativeObjAddr());`
6	
7	
8	`return mRgba;`
9	`}`
10	
11	`public native void FindFeatures(long matAddrGr, long matAddrRgba);`
12	

Penjelasan kode program diatas ialah:

Baris 1 menjelaskan bahwa implementasi ORB dimasukkan ke dalam prosedur `onCameraFrame`

Baris 5 merupakan prosedur yang dibuat untuk melakukan deteksi fitur menggunakan ORB. Baris ini menjelaskan bahwa terdapat 2 parameter masukkan yaitu citra RGB dan citra Grayscale.

Baris 11 merupakan prosedur yang berada dalam file *java* yang dideskripsikan sebagai sebuah prosedur native yang detilnya berada pada file cpp.

Penggunaan kode program yang lebih jelas dan terperinci terdapat pada lampiran file *KameraActivity.Java*.

Pada implementasi ORB dibutuhkan penulisan prosedur dalam bahasa C/C++, maka dari itu dibawah ini merupakan tampilan prosedur yang berada dalam file cpp.

Baris	Kode Program
1	`#include <opencv2/features2d/features2d.hpp>`
2	`#include <vector>`
3	
4	`using namespace std;`
5	`using namespace cv;`
6	
7	`extern "C" {`
8	
9	`JNIEXPORT void JNICALL`
10	`Java_com_ardipseptiadi_deteksihambatanberjalan_KameraActivity_`
11	`FindFeatures (JNIEnv*, jobject, jlong addrGray, jlong addrRgba)`
12	`{`
13	`    Mat& mGr = *(Mat*)addrGray;`
14	`    Mat& mRgb = *(Mat*)addrRgba;`
15	`    Mat descriptors_scene;`
16	`    vector<KeyPoint> keypoints_scene;`
17	
18	`    Ptr<ORB> detector = ORB::create();`
19	`    detector->detect(mGr, keypoints_scene);`
20	`    for( unsigned int i = 0; i < keypoints_scene.size(); i++ )`
21	`    {`
22	`        const KeyPoint& kp = keypoints_scene[i];`
23	`        circle(mRgb, Point(kp.pt.x, kp.pt.y), 10,`
24	`Scalar(255,0,0,255));`
25	`    }`
26	`}`

Penjelasan kode program diatas ialah:

Baris 9-12 merupakan deklarasi sebuah nama prosedur dalam file native cpp yang digunakan. Parameter masukan pun ditambahkan dengan parameter JNIENV* dan jobject

Baris 13 dan 14 menjelaskan bahwa inputan dari parameter didefinisikan ulang dengan tipe data Mat.

Baris 15 dan 16 merupakan definisi variabel baru.

Baris 18 merupakan sebuah definisi metode ORB yang memiliki tipe data Pointer yang dimasukan pada variabel detector.

Baris 19 merupakan implementasi metode deteksi fitur yang terdapat pada method `detect()` dengan parameter citra input dan output berupa vector keypoint.

Untuk memperlihatkan fitur yang telah terdeteksi maka pada baris 20-25 dilakukan pembuatan lingkaran pada setiap fitur yang ditemukan.

Penggunaan kode program yang lebih lengkap dapat dilihat dalam file native *jni_part.cpp*.

3.1.7.2.3 Ekstraksi Fitur

Setelah fitur dapat dideteksi, langkah berikutnya ialah diolah. Proses pengolahan melibatkan data dari tahap deteksi fitur dengan dilakukan filtrasi terhadap noise yang ada dan fitur yang dianggap tidak penting dan mengubahnya ke dalam *feature vector*. Output dalam tahap ini ialah kumpulan *feature vector*.

Metode yang digunakan untuk melakukan ekstraksi fitur ialah ORB sama seperti metode yang digunakan pada tahap deteksi fitur. Pada ekstraksi fitur, ORB dibangun berdasarkan metode BRIEF yang dapat mendeskripsi fitur pada citra.

Seperti halnya pada tahap deteksi fitur, implementasi ORB diterapkan dalam file berekstensi .cpp namun ada perbedaan *method* yang digunakan. Modul yang digunakan juga berasal dari *features2d*.

Berikut merupakan contoh implementasi metode ORB dengan modul *features2d* yang digunakan untuk membangun aplikasi ini:

Baris	Kode Program
1	`public Mat onCameraFrame(CvCameraViewFrame inputFrame) {`
2	
3	`mRgba = inputFrame.rgba();`
4	`mGray = inputFrame.gray();`
5	`FindFeatures(mGray.getNativeObjAddr(),mRgba.getNativeObjAddr());`
6	
7	
8	`return mRgba;`
9	`}`
10	
11	`public native void FindFeatures(long matAddrGr, long matAddrRgba);`
12	

Penjelasan kode program diatas ialah:

Baris 1 menjelaskan bahwa implementasi ORB dimasukkan ke dalam prosedur `onCameraFrame`

Baris 5 merupakan prosedur yang dibuat untuk melakukan deteksi fitur menggunakan ORB. Baris ini menjelaskan bahwa terdapat 2 parameter masukkan yaitu citra RGB dan citra Grayscale.

Baris 11 merupakan prosedur yang berada dalam file *java* yang dideskripsikan sebagai sebuah prosedur native yang detilnya berada pada file cpp.

Penggunaan kode program yang lebih jelas dan terperinci terdapat pada lampiran file *KameraActivity.Java*.

Seperti pada tahap deteksi fitur, implementasi ORB untuk ekstraksi fitur terdapat dalam file native cpp. Berikut merupakan implementasi metode untuk ekstraksi fitur:

Baris	Kode Program
1	`#include <opencv2/features2d/features2d.hpp>`
2	`#include <vector>`
3	
4	`using namespace std;`
5	`using namespace cv;`
6	
7	`extern "C" {`
8	`JNIEXPORT void JNICALL`
9	`Java_com_ardipseptiadi_deteksihambatanberjalan_KameraActivity_`
10	`FindFeatures (JNIEnv*, jobject, jlong addrGray, jlong addrRgba)`
11	`{`
12	`    Mat& mGr = *(Mat*)addrGray;`
13	`    Mat& mRgb = *(Mat*)addrRgba;`
14	`    Mat descriptors_scene;`
15	`    vector<KeyPoint> keypoints_scene;`
16	
17	`    Ptr<ORB> detector = ORB::create();`
18	`    detector->compute(mGr, keypoints_scene, descriptors_scene);`
19	`}`

Penjelasan kode program diatas ialah:

Penjelasan kode program diatas ialah:

Baris 9-10 merupakan deklarasi sebuah nama prosedur dalam file native cpp yang digunakan. Parameter masukan pun ditambahkan dengan parameter JNIENV* dan jobject

Baris 12 dan 13 menjelaskan bahwa inputan dari parameter didefinisikan ulang dengan tipe data Mat.

Baris 14 dan 15 merupakan definisi variabel baru.

Baris 17 merupakan sebuah definisi metode ORB yang memiliki tipe data Pointer yang dimasukan pada variabel detector.

Baris 18 merupakan implementasi metode ekstraksi fitur yang terdapat pada method `compute()` dengan parameter citra input dan output berupa vector keypoint.

Penggunaan kode program yang lebih lengkap dapat dilihat dalam file native *jni_part.cpp*.

3.1.7.3 Tahap Pencocokan Hambatan

Dalam tahap ini objek yang terdeteksi dilakukan pencocokan fitur terhadap citra yang telah didefinisikan sebelumnya. Pencocokan dapat diimplementasikan dengan berbagai cara dan pilihan metode yang sesuai dengan kecepatan dan akurasi. Keluaran yang dihasilkan oleh tahap ini ialah sebuah klausa kecocokan terhadap objek yang terdeteksi.

Metode yang digunakan dalam tahap ini ialah BruteForceMatcher. Metode ini merupakan sebuah *library* dalam OpenCV yang berguna untuk mencocokan fitur tetangga yang terdekat dengan citra yang telah didefinisikan sebelumnya.

Implementasi BruteForceMatcher terdapat pada modul yang sama dengan metode deteksi fitur yaitu *features2d*. Sama halnya seperti metode ORB,

BruteForceMatcher diterapkan dalam file native cpp. Berikut merupakan implementasi BruteForceMatcher untuk mencocokan fitur citra:

Baris	Kode Program
1	`if(!descriptors_scene.empty() && !descriptors_object.empty()){`
2	
3	`cv::BFMatcher matcher(cv::NORM_HAMMING);`
4	`std::vector<cv::DMatch> matches;`
5	
6	`matcher.match(descriptors_object,descriptors_scene,matches);`
7	
8	`double max_dist = 0; double min_dist = 100;`
9	
10	`for( int i = 0; i < descriptors_object.rows; i++ )`
11	`{`
12	`double dist = matches[i].distance;`
13	`if( dist < min_dist ) min_dist = dist;`
14	`if( dist > max_dist ) max_dist = dist;`
15	`}`
16	
17	`vector< cv::DMatch > good_matches;`
18	`for( int i = 0; i < descriptors_object.rows; i++ )`
19	`{`
20	`if( matches[i].distance < 3*min_dist )`
21	`{`
22	`good_matches.push_back(matches[i]);`
23	`}`
24	`}`
25	
26	`Mat img_matches;`
27	
28	
29	`vector<Point2f> obj;`
30	`vector<Point2f> scne;`
31	
32	`for( size_t i = 0; i < good_matches.size(); i++ )`
33	`{`
34	`//-- mengambil keypoint dari good_matches`
35	`obj.push_back( keypoints_object[`
36	`good_matches[i].queryIdx ].pt );`
37	`scene.push_back( keypoints_scene[`
38	`good_matches[i].trainIdx ].pt );`
39	`}`
40	`if(obj.size()>3){`
41	`Mat H = findHomography( obj, scne, RANSAC );`
42	`//-- mendapatkan corner fitur pada citra keypoint`
43	`vector<Point2f> obj_corners(4);`
44	`obj_corners[0] = cvPoint(0,0); obj_corners[1] =`
45	`cvPoint( grayObject.cols, 0 );`
46	`obj_corners[2] = cvPoint( grayObject.cols,`
47	`grayObject.rows ); obj_corners[3] = cvPoint( 0, grayObject.rows );`
48	`vector<Point2f> scene_corners(4);`
49	`perspectiveTransform( obj_corners, scene_corners, H);`

```
50              //-- Menggambarkan garis yang membentuk kotak
51              line( result, scene_corners[0] + Point2f(
52      grayObject.cols, 0), scene_corners[1] + Point2f( grayObject.cols,
53      0), Scalar(0, 255, 0), 4 );
54              line( result, scene_corners[1] + Point2f(
55      grayObject.cols, 0), scene_corners[2] + Point2f( grayObject.cols,
56      0), Scalar( 0, 255, 0), 4 );
57              line( result, scene_corners[2] + Point2f(
58      grayObject.cols, 0), scene_corners[3] + Point2f( grayObject.cols,
59      0), Scalar( 0, 255, 0), 4 );
60              line( result, scene_corners[3] + Point2f(
61      grayObject.cols, 0), scene_corners[0] + Point2f( grayObject.cols,
62      0), Scalar( 0, 255, 0), 4 );
63          }
64          }
```

Penjelasan kode program diatas ialah:

Baris 1 menjelaskan bahwa metode BruteForceMatcher hanya akan dijalankan jika deskriptor kedua citra tidak kosong.

Baris 3 merupakan definisi tipe data untuk metode BruteForceMatcher terhadap variabel matcher.

Baris 4 merupakan define vector Dmatch untuk variabel matches.

Baris 6 berisikan metode `match()` yang merupakan *method* yang dimiliki oleh BruteForceMatcher untuk melakukan pencocokan citra. Adapun parameter masukan yang berasal dari nilai deskriptor kedua citra serta keluaran berupa vector yang disimpan pada variabel matches.

Baris 8-15 merupakan definisi jarak untuk mendapatkan nilai kecocokan yang terdapat variabel matches

Baris 17-24 merupakan seleksi jarak yang terpilih yang dianggap fitur yang cocok pada citra.

Baris 26-39 merupakan langkah mengambil keypoint fitur pada citra yang dianggap cocok.

Baris 41 merupakan implementasi metode Homography.

Baris 43-48 merupakan langkah untuk mengambil nilai *corner* pada citra yang telah cocok.

Baris 49 merupakan penerapan transformasi perspektif.

Baris 61-63 merupakan langkah untuk menggambar sebuah persegi untuk memperlihatkan citra yang cocok.

Penggunaan kode program yang lebih lengkap dapat dilihat dalam file native *jni_part.cpp*.

3.1.7.4 Output Informasi

Output Informasi merupakan tahap akhir dalam penelitian ini. Setelah suatu objek terklasifikasi, maka informasi akan diberikan kepada tunanetra melalui suara. Agar informasi yang diberikan terhadap tunanetra dapat berguna sebagaimana mestinya, output informasi keberadaan hambatan diberikan melalui suara.

3.1.8 Analisis Spesifikasi Kebutuhan Perangkat Lunak (SKPL)

Analisis spesifikasi kebutuhan perangkat lunak merupakan spesifikasi dari perangkat lunak yang akan dikembangkan. Analisis SKPL dibutuhkan untuk menentukan spesifikasi kebutuhan sistem dari sisi fungsional dan non fungsional.

3.1.8.1 Kebutuhan Non – Fungsional

Analisis spesifikasi kebutuhan non fungsional berisi batasan – batasan dari layanan – layanan dan fungsi – fungsi dari sebuah sistem, seperti batasan waktu, batasan dari pengembangan proses, serta batasan pengguna.

Tabel 3.2 Spesifikasi Kebutuhan Non - Fungsional

Nomor	Spesifikasi Kebutuhan Non Fungsional
SKPL-NF-001	Sistem ini dibangun pada sistem operasi Android.
SKPL-NF-002	Sistem ini dapat digunakan oleh penyandang tunanetra.

3.1.8.2 Kebutuhan Fungsional

Analisis spesifikasi kebutuhan fungsional berisi pernyataan dari layanan sistem yang harus disediakan, bagaimana sistem harus berjalan terhadap input tertentu dan bagaimana sistem harus berperilaku dalam situasi tertentu. Analisis spesifikasi kebutuhan fungsional menjelaskan fungsionalitas dari sistem serta menjelaskan layanan dari sistem secara detail.

Tabel 3.3 Kebutuhan Fungsional

User Requirement	
Nomor	**Spesifikasi Kebutuhan Fungsional**
SKPL-F-001	User dapat mengetahui keberadaan hambatan di depannya
SKPL-F-002	User dapat menghindari hambatan yang mungkin dapat membahayakan
System Requirement	
Nomor	**Spesifikasi Kebutuhan Fungsional**
SKPL-F1-001	User mendapatkan peringatan bahaya
SKPL-F2-001	User mendapatkan informasi arah

3.1.9 Analisis Kebutuhan Non Fungsional

Analisis kebutuhan non fungsional terdiri atas kebutuhan perangkat keras, perangkat lunak dan pengguna.

3.1.9.1 Analisis Kebutuhan Perangkat Keras

Kebutuhan perangkat keras untuk pengembangan aplikasi ini ialah dengan spesifikasi sebagai berikut:

Tabel 3.4 Kebutuhan Perangkat Keras

No.	Perangkat Keras	Spesifikasi
1	Perangkat Komputer (Laptop)	Prosesor Intel Core i3 @2.40 Ghz
		Memory RAM 4GB
		Harddisk 1TB
		VGA Shared 2GB
		LCD 14 inci
2	Android	Kamera 2MP
		Memory RAM 1GB
		Internal Memory 8GB
		Prosesor QuadCore @1.2 Ghz

3.1.9.2 Analisis Kebutuhan Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan program aplikasi diantaranya:

Tabel 3.5 Kebutuhan Perangkat Lunak

No.	Perangkat Lunak	Spesifikasi
1	Sistem Operasi Perangkat Komputer (Laptop)	Windows 7 Ultimate Edition
2	Sistem Operasi Android	Android v4.4 Kitkat
3	Tools IDE	Android Studio v2.1.0
4	Library OpenCV	OpenCV 3.1.0

3.1.9.3 Analisis Pengguna

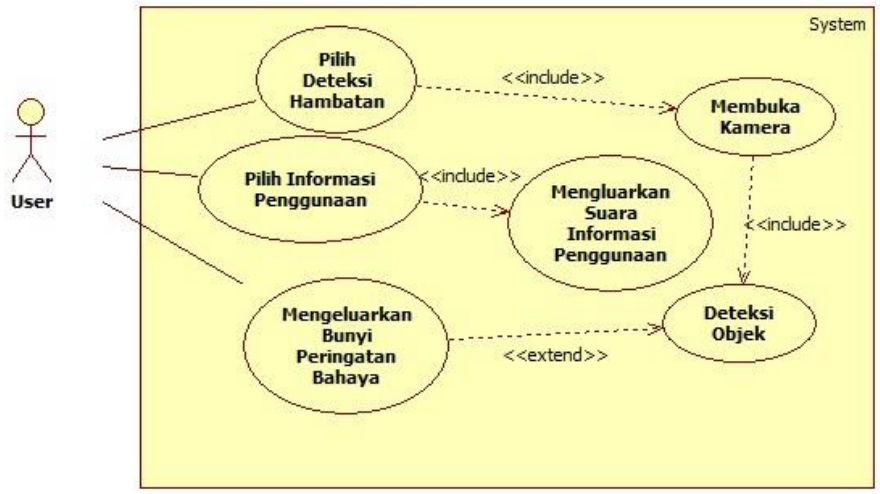
Aplikasi yang akan dibangun ditujukan bagi penyandang tunanetra, khususnya dapat menggunakan serta memiliki *smartphone* Android. Tidak hanya tunanetra, seseorang yang mengalami gangguan sementara pada penglihatan pun dapat menggunakan aplikasi ini.

3.1.10 Analisis Kebutuhan Fungsional

Kebutuhan fungsional terdiri atas beberapa diagram UML, diantaranya: diagram *use case*, diagram *activity*, diagram *class* dan diagram *sequence*. Diagram tersebut dipergunakan sebagai desain rancangan untuk mempermudah proses pengembangan aplikasi khususnya untuk aplikasi yang dikembangkan menggunakan program berorientasi objek.

3.1.10.1 *Use Case Diagram*

Use Case diagram merupakan gambaran skenario dari interaksi antara *user* dengan sistem. Sebuah diagram *Use Case* yang menggambarkan hubungan antara actor dan kegiatan yang dapat dilakukannya terhadap aplikasi. Didalam sistem terdapat pengguna yaitu pemakai aplikasi. Peran aktor yang dapat terlihat pada diagram *Use Case* pada gambar:



Gambar 3.1 Use Case Diagram

1. Definisi Aktor

Definisi Aktor berfungsi untuk menjelaskan aktor yang tercantum pada *Use Case Diagram.*

Tabel 3.6 Definisi Aktor

No.	Actor	Deskripsi
1	User	Orang yang menggunakan aplikasi pada sistem, yaitu pengguna yang dalam penelitian ini seorang tunanetra.

2. Definisi Use Case

Definisi use case berfungsi untuk menjelaskan fungsi *Use Case* yang terdapat pada *Use Case Diagram*.

Tabel 3.7 Definisi Use Case

NO	*Use Case*	Deskripsi
1	Pilih Deteksi Hambatan	Fungsionalitas untuk memulai mendeteksi hambatan
2	Membuka Kamera	Fungsionalitas untuk membuka kamera pada perangkat
3	Deteksi objek	Fungsionalitas untuk mendeteksi objek
4	Pilih Informasi Penggunaan	Fungsionalitas untuk menampilkan atau memberi informasi penggunaan
5	Mengeluarkan Suara Informasi Penggunaan	Fungsionalitas untuk mengeluarkan suara informasi penggunaan
6	Mengeluarkan Bunyi Peringatan Bahaya	Fungsionalitas untuk menampilkan atau memberi peringatan adanya hambatan

3. Skenario Use Case

Skenario *Use Case* mendeskripsikan urutan langkah – langkah dalam proses bisnis, baik yang dilakukan oleh aktor. Berdasarkan *use case* yang ditampilkan pada gambar 3.1 maka scenario *use case* akan dijelaskan sebagai berikut:

a. Skenario *Use Case* Pilih Deteksi Hambatan

Skenario *use case* Pilih Deteksi Hambatan dijelaskan pada tabel 3.7 berikut:

Tabel 3.8 Usecase Deteksi Hambatan

Use Case Name	**Pilih Deteksi Hambatan**	
Related requirement	Membuka Kamera	
Goals in Context	User ingin memulai mendeteksi hambatan yang ditemui	
Preconditions	Sistem menampilkan menu utama	
Successful End Condition	Sistem berhasil memulai Deteksi Hambatan	
Failed End Condition	Sistem gagal memulai Deteksi Hambatan	
Primary Actors	User	
Secondary Actors	-	
Trigger	User memilih menu “Mulai Deteksi Hambatan”	
Main Flow	*Step*	*Action*
	1	Sistem menampilkan menu utama
	2	User memilih menu “Mulai Deteksi Hambatan”
	3	Sistem berhasil memulai Deteksi Hambatan
Extensions	*Step*	*Branching action*
	3.1	Sistem gagal memulai Deteksi Hambatan

b. Skenario *Use Case* Membuka Kamera

Skenario *use case* Membuka Kamera dijelaskan pada tabel 3.8 berikut:

Tabel 3.9 Usecase Membuka Kamera

Use Case Name	**Membuka Kamera**
Related requirement	Pilih Deteksi Hambatan
Goals in Context	Sistem ingin membuka kamera
Preconditions	Sistem memulai Deteksi Hambatan
Successful End Condition	Sistem berhasil membuka kamera

Failed End Condition	Sistem gagal membuka kamera	
Primary Actors	User	
Secondary Actors	-	
Trigger	Sistem berhasil memulai Deteksi Hambatan	
Main Flow	*Step*	*Action*
	1	Sistem memulai Deteksi Hambatan
	2	Sistem membuka kamera
	3	Sistem berhasil membuka kamera
Extensions	*Step*	*Branching action*
	3.1	Sistem gagal membuka kamera

c. Skenario *Use Case* Deteksi Objek

Skenario *use case* Deteksi Objek dijelaskan pada tabel 3.9 berikut:

Tabel 3.10 Usecase Deteksi Objek

Use Case Name	**Deteksi Objek**	
Related requirement	Membuka Kamera	
Goals in Context	Sistem dapat memulai mendeteksi objek	
Preconditions	Sistem membuka kamera	
Successful End Condition	Sistem mendeteksi objek	
Failed End Condition	Sistem gagal mendeteksi objek	
Primary Actors	Pengguna	
Secondary Actors	-	
Trigger	Pengguna menemui hambatan	
Main Flow	*Step*	*Action*
	1	Sistem membuka kamera
	2	Pengguna menemui hambatan
	3	Sistem berhasil mendeteksi objek
Extensions	*Step*	*Branching Action*
	3.1	Sistem gagal mendeteksi objek

d. Skenario *Use Case* Pilih Informasi Penggunaan

Skenario *use case* Pilih Informasi Penggunaan dijelaskan pada tabel 3.10 berikut:

Tabel 3.11 Usecase Informasi Penggunaan

Use Case Name	**Pilih Informasi Penggunaan**	
Related requirement	Tampil Informasi Penggunaan	
Goals in Context	Pengguna mendapatkan informasi penggunaan aplikasi	
Preconditions	Sistem menampilkan halaman utama	
Successful End Condition	Sistem berhasil memilih Informasi Penggunaan	
Failed End Condition	Sistem gagal memilih Informasi Penggunaan	
Primary Actors	Pengguna	
Secondary Actors	-	
Trigger	Pengguna memilih menu "Informasi Penggunaan"	
Main Flow	*Step*	*Action*
	1	Sistem menampilkan halaman utama
	2	Pengguna memilih menu "Informasi Penggunaan"
	3	Sistem berhasil memilih Informasi Penggunaan
Extensions	*Step*	*Branching action*
	3.1	Sistem gagal memilih Informasi Penggunaan

e. Skenario *Use Case* Mengeluarkan Suara Informasi Penggunaan

Skenario *use case* Tampil Informasi Arah dijelaskan pada tabel 3.11 berikut:

Tabel 3.12 Usecase Mengeluarkan Suara Informasi Penggunaan

Use Case Name	**Mengeluarkan Suara Informasi Penggunaan**	
Related requirement	Pilih Informasi Penggunaan	
Goals in Context	Mengeluarkan suara informasi penggunaan	
Preconditions	Sistem berada di menu utama	
Successful End Condition	Sistem mengeluarkan suara	
Failed End Condition	Sistem tidak mengeluarkan suara	
Primary Actors	Pengguna	
Secondary Actors	-	
Trigger	User memilih tombol “Informasi Penggunaan”	
Main Flow	*Step*	*Action*
	1	Sistem menampilkan halaman utama
	2	User memilih tombol “Informasi Penggunaan”
	3	Sistem berhasil mengeluarkan suara
Extensions	*Step*	*Branching action*
	3.1	Sistem tidak berhasil mengeluarkan suara

f. Skenario *Use Case* Tampil Peringatan Bahaya

Skenario *use case* Tampil Peringatan Bahaya dijelaskan pada tabel 3.12 berikut:

Tabel 3.13 Usecase Mengeluarkan Bunyi Peringatan Bahaya

Use Case Name	**Mengeluarkan Bunyi Peringatan Bahaya**
Related requirement	Deteksi Objek
Goals in Context	Sistem memberikan informasi peringatan bahaya kepada user untuk menghindari ancaman
Preconditions	Sistem berhasil mendeteksi objek
Successful End Condition	Sistem berhasil memberikan peringatan bahaya

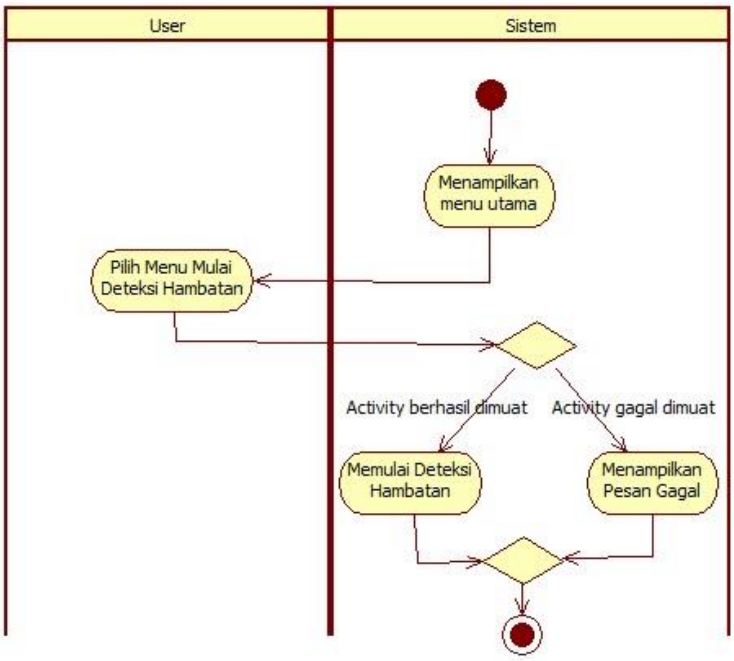
Failed End Condition	Sistem gagal memberikan peringatan bahaya	
Primary Actors	Pengguna	
Secondary Actors	-	
Trigger	Sistem mendeteksi objek yang dianggap berbahaya	
Main Flow	*Step*	*Action*
	1	Sistem mendeteksi objek
	2	Sistem mendeteksi objek yang berbahaya
	3	Sistem berhasil menampilkan peringatan berbahaya
Extensions	*Step*	*Branching action*
	3.1	Sistem gagal menampilkan peringatan berbahaya

3.1.10.2 *Activity Diagram*

Activity diagram merupakan diagram yang memodelkan aliran kerja atau *workflow* dari urutan aktivitas dalam suatu proses yang mengacu pada *use case diagram* yang ada.

1. *Activity Diagram* Pilih Deteksi Hambatan

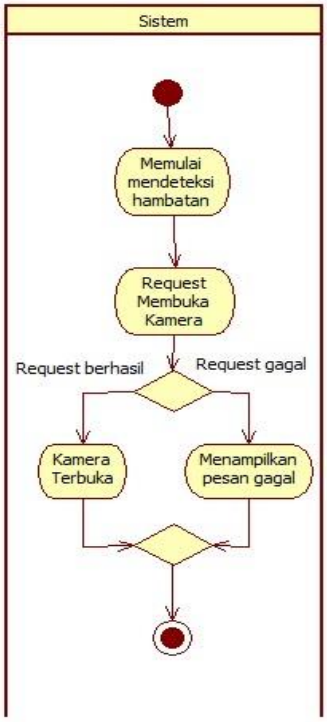
Activity diagram pilih deteksi hambatan menjelaskan aliran kerja aktor user pada saat memilih deteksi hambatan. *Activity diagram* pilih deteksi hambatan dapat dilihat pada Gambar 3.2.



Gambar 3.2 *Activity Diagram* Pilih Deteksi Hambatan

2. *Activity Diagram* Membuka Kamera

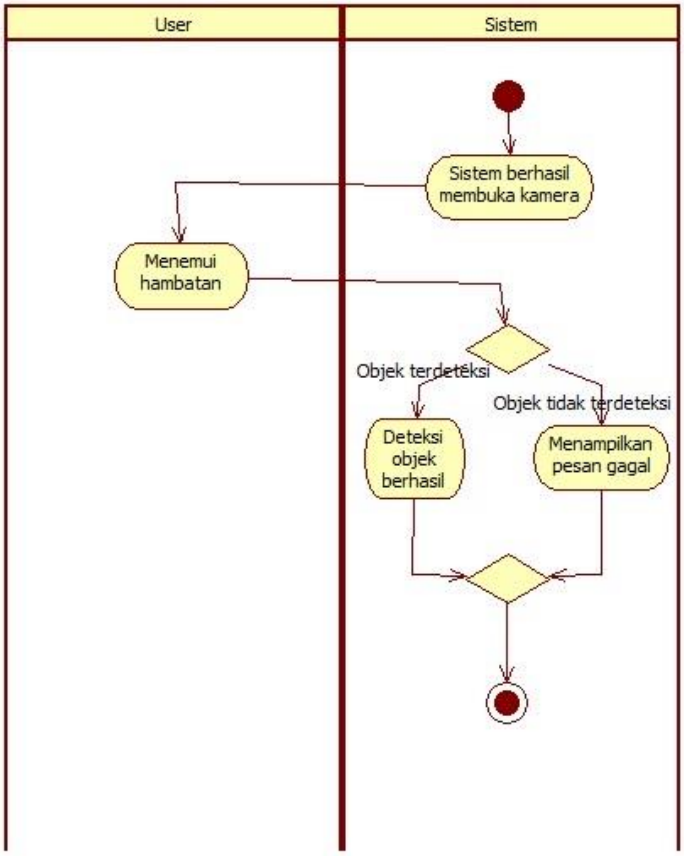
Activity diagram membuka kamera menjelaskan aliran kerja sistem pada saat membuka kamera. *Activity diagram* membuka kamera dapat dilihat pada Gambar 3.3.



Gambar 3.3 *Activity Diagram* Membuka Kamera

3. *Activity Diagram* Deteksi Objek

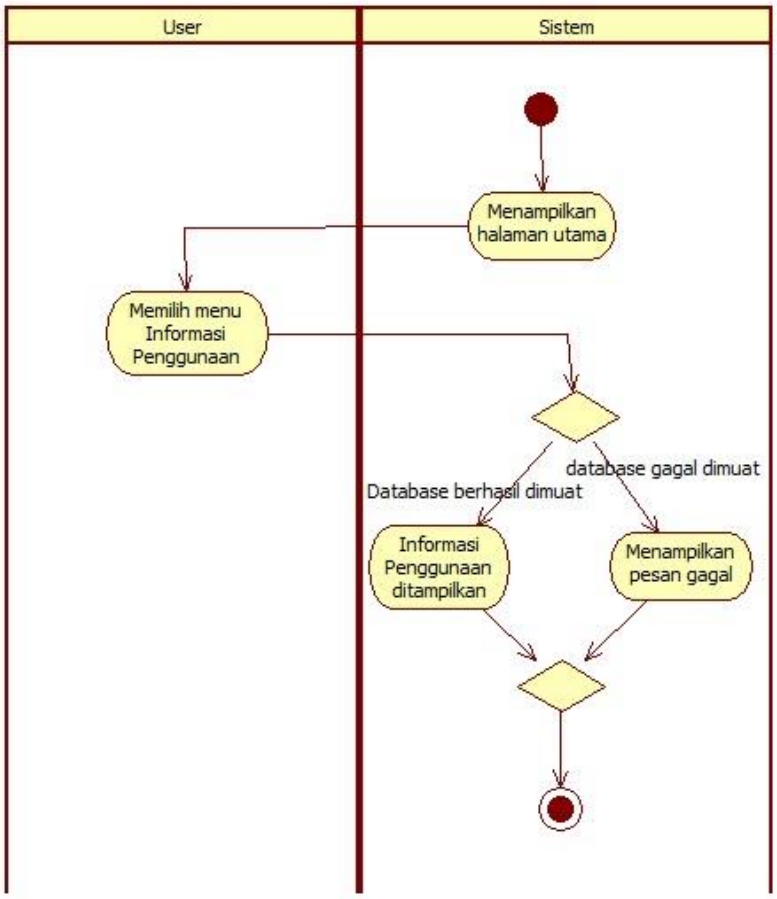
Activity diagram Deteksi Objek menjelaskan aliran kerja sistem pada saat mendeteksi objek. *Activity diagram* deteksi objek dapat dilihat pada Gambar 3.4.



Gambar 3.4 *Activity Diagram* Deteksi Objek

4. *Activity Diagram* Tampil Informasi Penggunaan

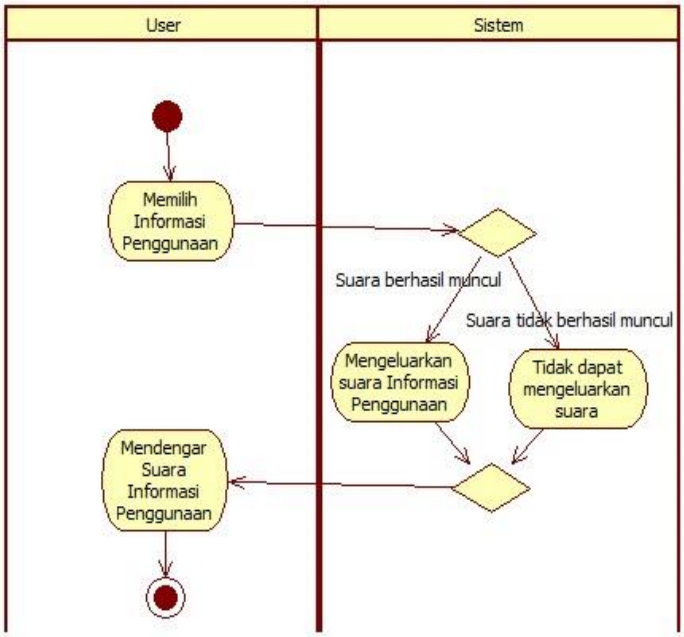
Activity diagram tampil informasi penggunaan menjelaskan aliran kerja aktor user pada saat menampilkan informasi penggunaan. *Activity diagram* tampil informasi penggunaan dapat dilihat pada Gambar 3.5.



Gambar 3.5 *Activity Diagram* Tampil Informasi Penggunaan

5. *Activity Diagram* Mengeluarkan Suara Informasi Penggunaan.

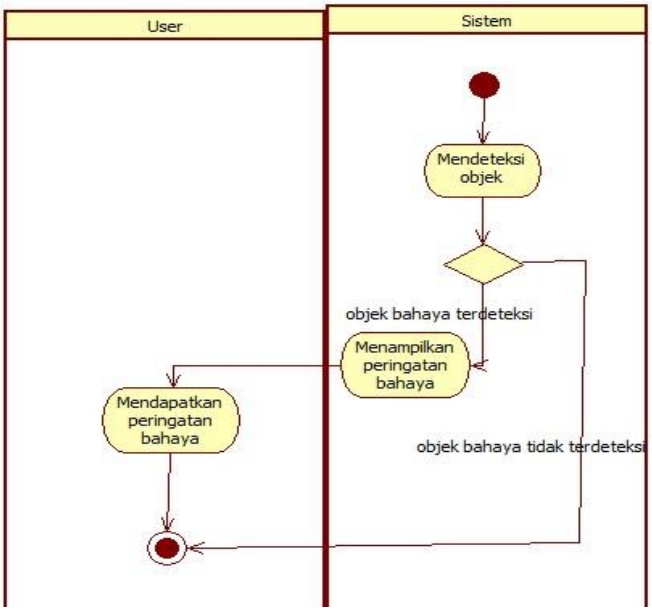
Activity diagram tampil informasi arah menjelaskan aliran kerja aktor user pada saat menampilkan informasi arah. *Activity diagram* tampil informasi arah dapat dilihat pada Gambar 3.6.



Gambar 3.6 *Activity Diagram* Mengeluarkan Suara Informasi Penggunaan

6. *Activity Diagram* Mengeluarkan Bunyi Peringatan Bahaya

Activity diagram pilih deteksi hambatan menjelaskan aliran kerja sistem pada saat menampilkan peringatan bahaya. *Activity diagram* tampil informasi arah dapat dilihat pada Gambar 3.7.



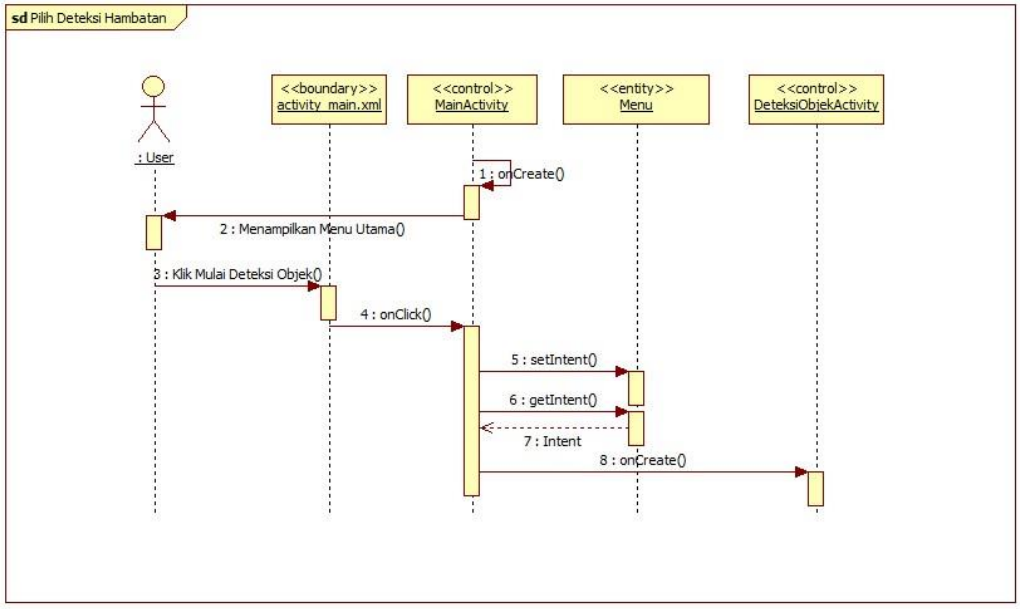
Gambar 3.7 *Activity Diagram* Mengeluarkan Bunyi Peringatan Bahaya

3.1.10.3 *Sequence Diagram*

Sequence diagram merupakan gambaran interaksi antar masing – masing objek pada setiap *use case* dalam urutan waktu. Interaksi ini berupa pengiriman serangkaian data antar objek – objek yang saling berinteraksi.

1. *Sequence Diagram* Pilih Deteksi Hambatan

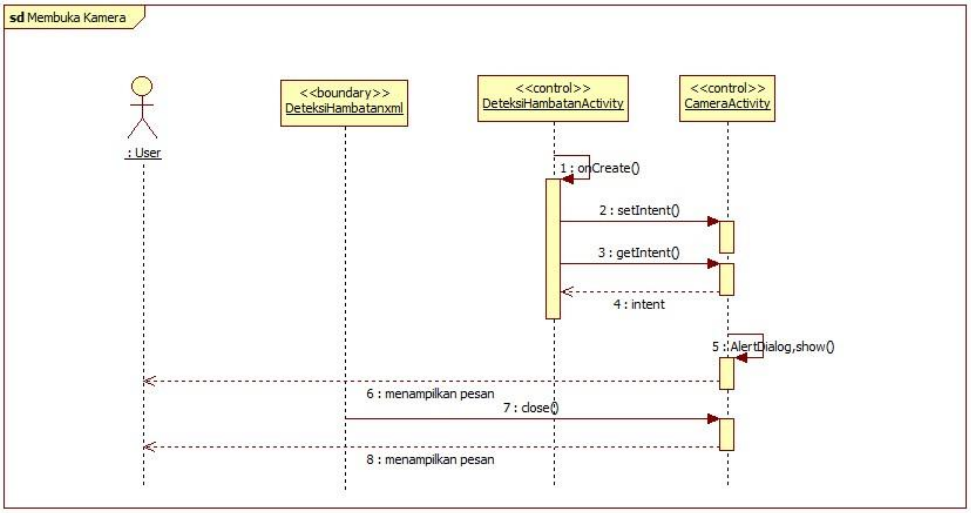
Sequence diagram pilih deteksi hambatan dapat dilihat pada Gambar 3.8.



Gambar 3.8 *Sequence Diagram* Pilih Deteksi Hambatan

2. *Sequence Diagram* Membuka Kamera

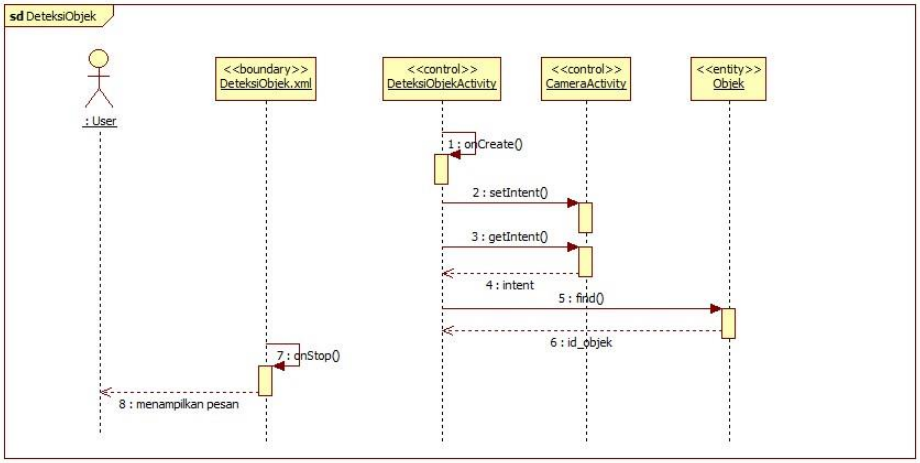
Sequence diagram membuka kamera dapat dilihat pada Gambar 3.9.



Gambar 3.9 *Sequence Diagram* Membuka Kamera

3. *Sequence Diagram* Deteksi Objek

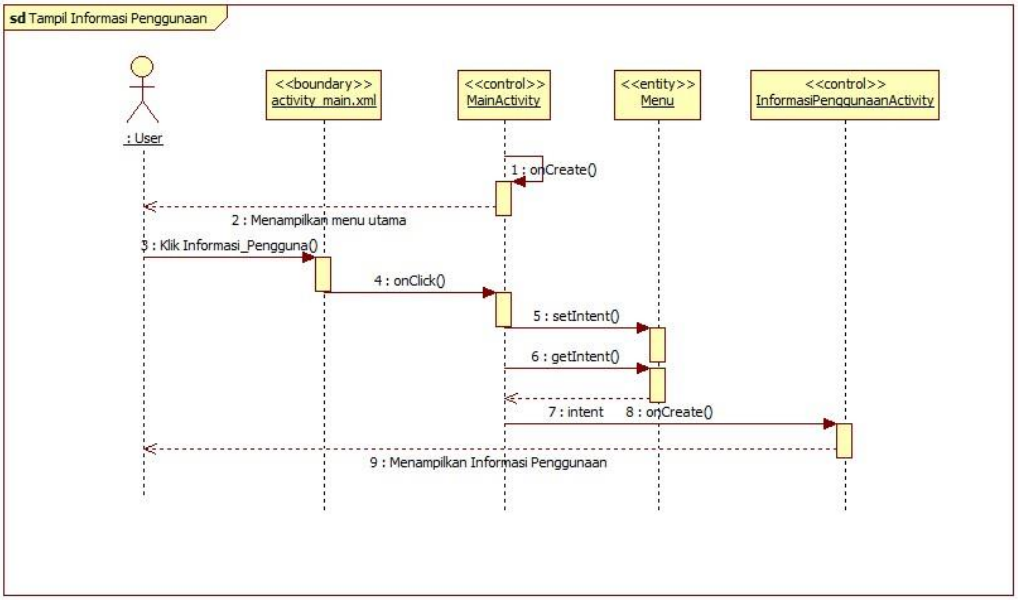
Sequence diagram deteksi objek dapat dilihat pada Gambar 3.10.



Gambar 3.10 *Sequence Diagram* Deteksi Objek

4. *Sequence Diagram* Tampil Informasi Penggunaan

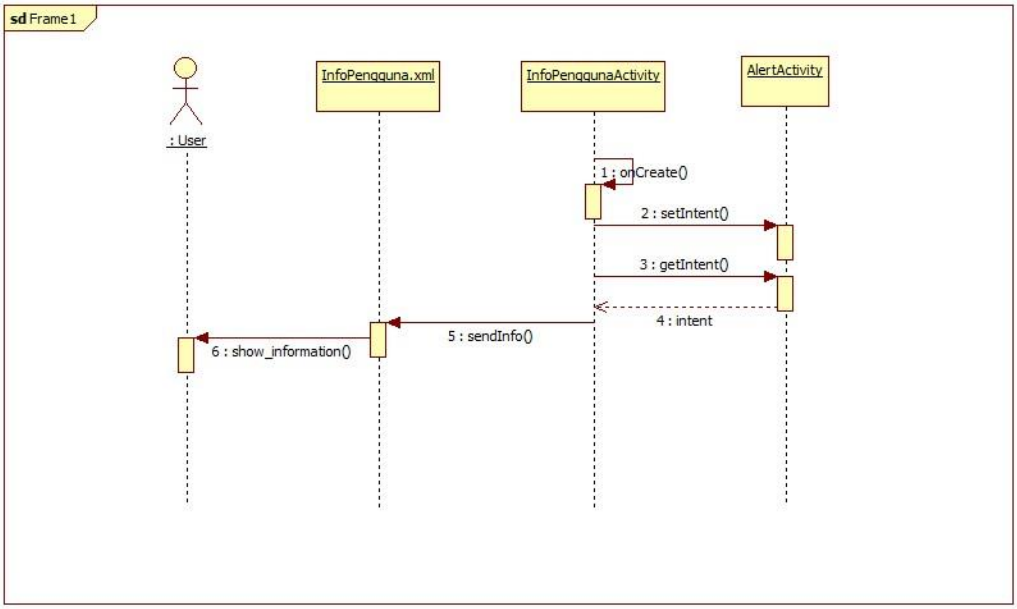
Sequence diagram tampil informasi penggunaan dapat dilihat pada Gambar 3.11.



Gambar 3.10 *Sequence Diagram* Tampil Informasi Penggunaan

5. *Sequence Diagram* Mengeluarkan Suara Informasi Penggunaan

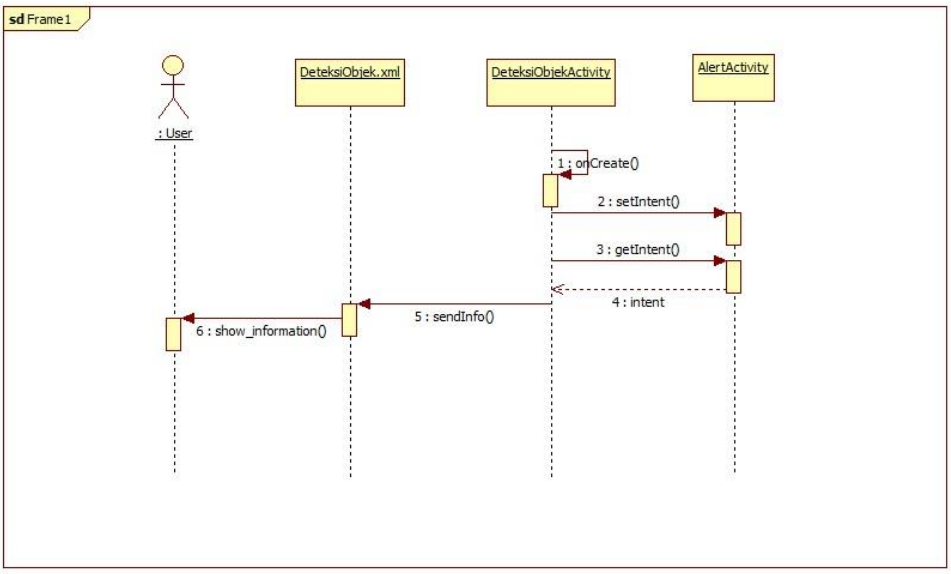
Sequence diagram Mengeluarkan Suara Informasi Penggunaan dapat dilihat pada Gambar 3.11.



Gambar 3.11 *Sequence Diagram* Mengeluarkan Suara Informasi Penggunaan

6. *Sequence Diagram* Mengeluarkan Bunyi Peringatan Bahaya

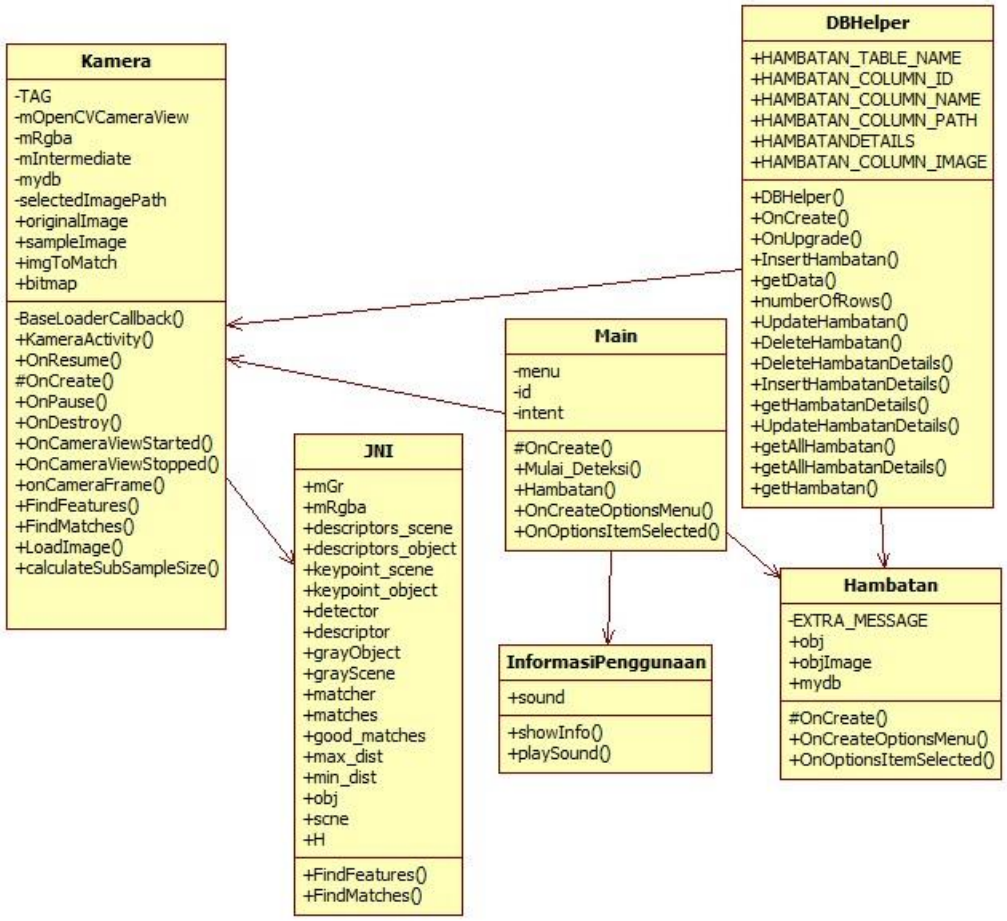
Sequence diagram Mengeluarkan Bunyi Peringatan Bahaya dapat dilihat pada Gambar 3.12.



Gambar 3.12 *Sequence Diagram* Mengeluarkan Bunyi Peringatan Bahaya

3.1.10.4 *Class Diagram*

Adapun gambaran *class diagram* dari aplikasi "Deteksi Hambatan Berjalan" yang dibangun dapat dilihat seperti berikut:



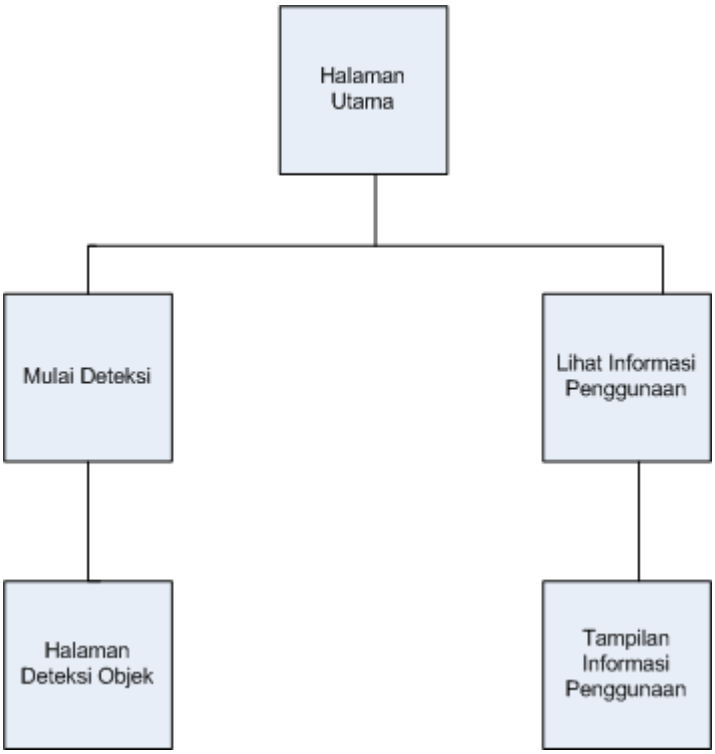
Gambar 3.13 *Class Diagram* Aplikasi "Deteksi Hambatan Berjalan"

3.2 Perancangan Sistem

Sistem aplikasi android yang akan dibangun memiliki fungsi utama untuk melakukan proses pendeteksian *obstacle* yang dapat menghambat tunanetra ketika berjalan. Untuk dapat mendeteksi *obstacle* sistem mengambil citra secara *real time* melalui kamera, lalu dideteksi objek yang terdapat pada citra, diklasifikasikan menurut tingkat bahaya dan sistem memberi informasi kepada tunanetra.

3.2.1 Perancangan Struktur Menu

Struktur menu merupakan sebuah bentuk umum rancangan program yang bertujuan untuk membantu pengguna dalam menyelesaikan sebuah tugas. Struktur menu untuk aplikasi "Deteksi Hambatan Berjalan" ialah sebagai berikut:



Gambar 3.16 Struktur Menu Aplikasi "Deteksi Hambatan Berjalan"

3.2.2 Perancangan Antarmuka

Perancangan antarmuka merupakan sebuah sketsa penggambaran dari penggabungan elemen – elemen yang dibutuhkan untuk mewakili keadaan sebenarnya dari aplikasi yang dibangun. Berikut adalah gambar – gambar dari perancangan antarmuka untuk aplikasi deteksi hambatan berjalan:

3.2.2.1 Perancangan *Form*

Perancangan *form* mendeskripsikan rencana tampilan dari setiap *form* yang akan digunakan pada tampilan aplikasi "Deteksi Hambatan Berjalan". Berikut gambaran perancangan *form* aplikasi yang dibuat:

1. Perancangan *Form* Halaman Utama

Perancangan *form* halaman utama merupakan kumpulan menu yang akan muncul pertama kali pada saat pengguna membuka aplikasi. Adapun perancangan *form* menu utama dapat dijelaskan sebagai berikut:

Tabel 3.12 Perancangan Antarmuka Halaman Utama

<table>
<tr><td colspan="2">A01</td></tr>
<tr><td>
</td><td>- Klik menu “Mulai Deteksi Objek” untuk menuju ke A02.
- Klik menu “Informasi Penggunaan” untuk menuju ke A03</td></tr>
<tr><td colspan="2">- Ukuran layar menyesuaikan
- Font, ukuran, dan warna tulisan menyesuaikan</td></tr>
</table>

2. Perancangan *Form* Deteksi Objek

Perancangan *form* deteksi objek merupakan rancangan fungsional yang dibuat untuk mendeteksi objek. *Form* ini akan muncul setelah user memilih menu “Mulai Deteksi Objek”. Adapun perancangan *form* deteksi objek seperti berikut:

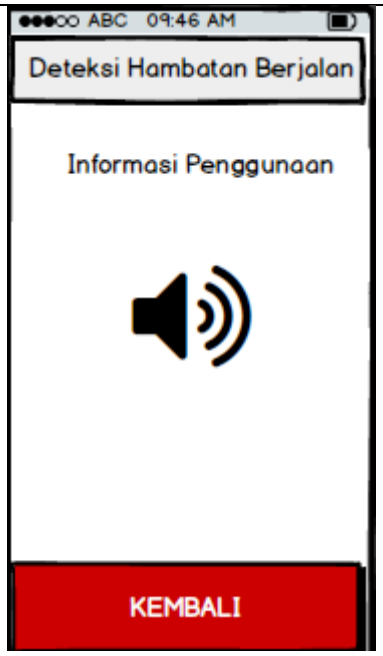
Tabel 3.13 Perancangan Antarmuka Deteksi Objek

<table>
<tr><td colspan="2">A02</td></tr>
<tr><td>
</td><td>- Jika objek terdeteksi maka akan memberikan informasi berupa suara
- Klik tombol “Berhenti” untuk kembali ke A01</td></tr>
<tr><td colspan="2">- Ukuran layar menyesuaikan
- Font, ukuran, dan warna tulisan menyesuaikan</td></tr>
</table>

3. Perancangan *Form* Informasi Penggunaan

Perancangan *form* informasi penggunaan merupakan rancangan menu yang berisi informasi cara penggunaan aplikasi “Deteksi Hambatan Berjalan”. *Form* ini akan muncul setelah user memilih menu “Informasi Penggunaan”. Adapun perancangan *form* informasi penggunaan seperti berikut:

Tabel 3.14 Perancangan Antarmuka Informasi Pengguna

<table>
<tr><td colspan="2">A03</td></tr>
<tr><td>
</td><td>- Ketika layar disentuh atau form ini mulai terbuka, maka akan muncul suara informasi penggunaan.
- Klik tombol “Berhenti” untuk kembali ke A01</td></tr>
<tr><td colspan="2">- Ukuran layar menyesuaikan
- Font, ukuran, dan warna tulisan menyesuaikan</td></tr>
</table>

3.2.2.2 Perancangan Pesan

Perancangan pesan mendeskripsikan rencana tampilan dari setiap pesan yang digunakan pada aplikasi “Deteksi Hambatan Berjalan”. Berikut gambaran perancangan pesan pada aplikasi yang dibuat:

1. Perancangan Pesan Gagal Membuka Kamera

Pesan gagal membuka kamera berisi informasi bahwa sistem gagal membuka kamera karena terjadi suatu *crash*. Berikut gambaran pesan gagal membuka kamera:

Tabel 3.15 Perancangan Pesan Gagal Membuka Kamera

<table>
<tr><td colspan="2">P01</td></tr>
<tr><td>
</td><td>- Jika Form ini muncul akan mengeluarkan suara pesan kesalahan
- Klik Kembali untuk menuju A01</td></tr>
<tr><td colspan="2">- Ukuran pesan menyesuaikan
- Font, ukuran, dan warna tulisan menyesuaikan</td></tr>
</table>

2. Perancangan Pesan Gagal Menampilkan Informasi Penggunaan

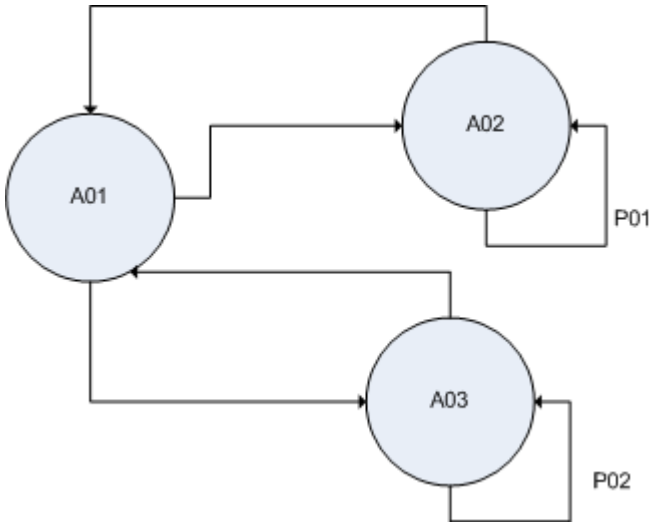
Pesan gagal menampilkan informasi penggunaan berisi informasi bahwa sistem gagal menampilkan informasi karena terjadi suatu crash. Berikut gambaran pesan gagal menampilkan informasi penggunaan:

Tabel 3.16 Perancangan Pesan Gagal Membuka Informasi Penggunaan

<table>
<tr><td colspan="2">P02</td></tr>
<tr><td>
</td><td>- Jika Form ini muncul akan mengeluarkan suara pesan kesalahan
- Klik Kembali untuk menuju A02</td></tr>
<tr><td colspan="2">- Ukuran pesan menyesuaikan
- Font, ukuran, dan warna tulisan menyesuaikan</td></tr>
</table>

3.2.3 Jaringan Semantik

Jaringan semantik merupakan gambaran pengetahuan grafis yang menunjukkan hubungan antar objek (lingkaran) dan garis yang menggambarkan formasi antar objek tersebut.



Gambar 3.17 Jaringan Semantik Aplikasi "Deteksi Hambatan Berjalan"