OBJECT ORIENTED ANALYSIS AND DESIGN PROJECT

TITLE: IOT BASED ANTI-THEFT DETECTION AND ALARM SYSTEM

TEAM NAME: BUQ SQUASHERS

TEAM MEMBERS:

REGISTER NUMBER	NAME
2019103523	HARIPRIYA K
2019103526	HEMAVARSHINI C
2019103553	POOJASRI S

Table of contents

1.Abstract:

- Now-a-days, Security has become the most challenging task.
- Everyone wants safety but in present scenario, nothing is safe not even in their own houses. We generally lock houses when going out of the house.
- However, simply locking your home isn't sufficient; a home security system should keep track of activities and report them to the homeowner, and work according to the owner's instructions.

2.Description:

- This system monitors the entire floor movement.
- One single step anywhere on the floor is tracked and the user is alarmed through mail over IOT.
- In this system secure flooring tile is connected with IOT, when the system is turned on.
- Whenever the thief enters in the house, and steps on the floor immediately it is sensed by the sensor which passes the signal to raspberry pi controller.
- The controller in turn processes it to be a valid signal and then moves the camera to the area where movement was detected to capture face image.
- It then undergoes a face recognition technique where it compares the processed face with the images stored in the database to verify who the person is.
- If the face is unrecognized then the system transmits it over the internet for the home owner to check the image.

3.Intended Audience:

- ✓ This system is useful for all kinds of people to keep track of all the activities of their homes and to notify the home owners when a thief tries to enter the home and thereby helps to take actions on time.
- ✓ The intended audience would include the emergency services like the control room police station.

4.Proposed Mini Case Study Statement:

- To design a IoT based anti-theft detection and alarm system to detect the presence of thief trying to enter the home.
- And to alert the home owners so that an immediate action would be taken by them to reduce the possibilities of thefts.

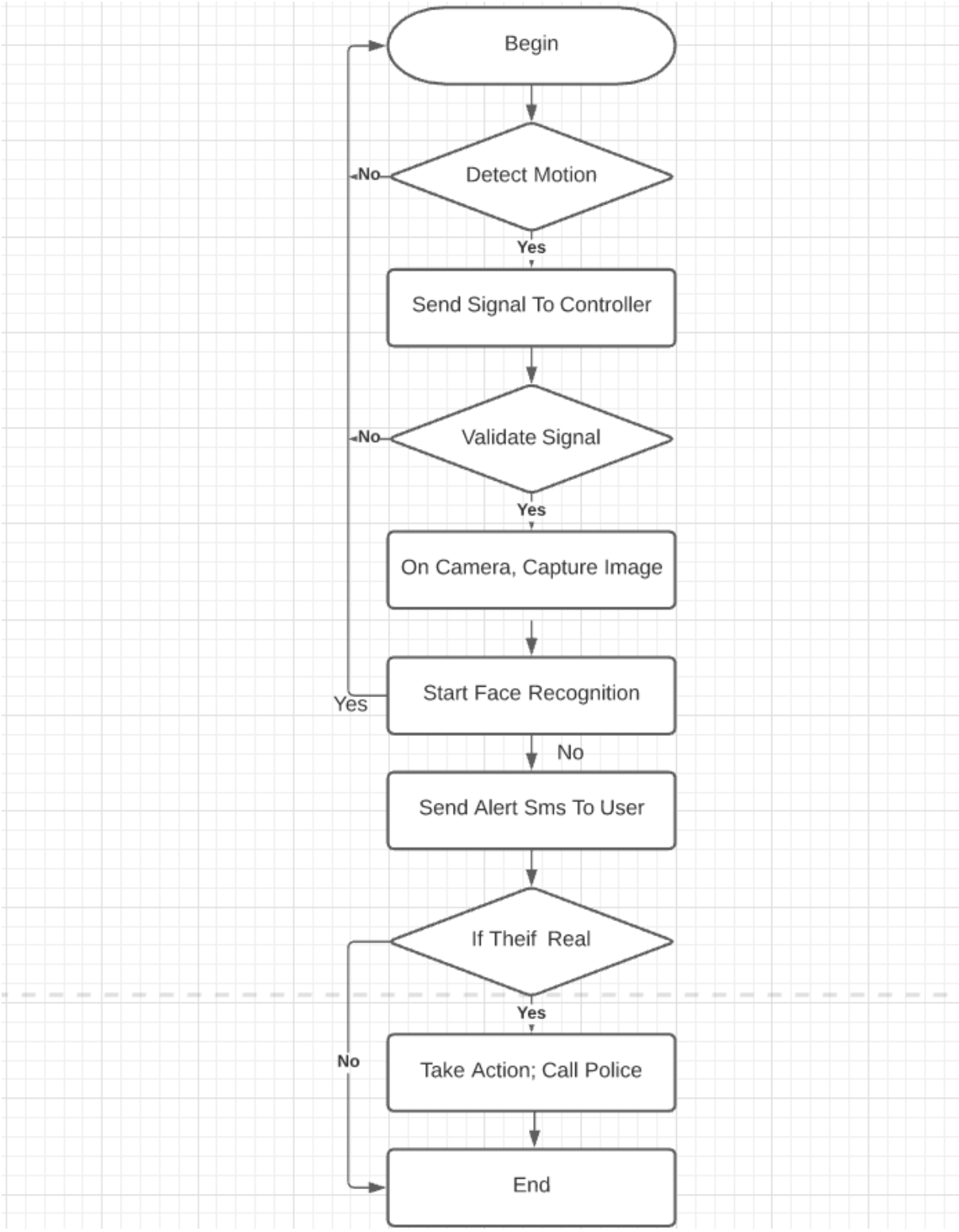
5.Benefits of the system:

- This system is capable of distinguishing between human and animal intrusion using sensor for body temperature detection.
- It uses alarm system which uses to alert the owner by making sound.
- It is convenient to use relatively free from false alarms and does not require frequent user action to arm and disarm the system.

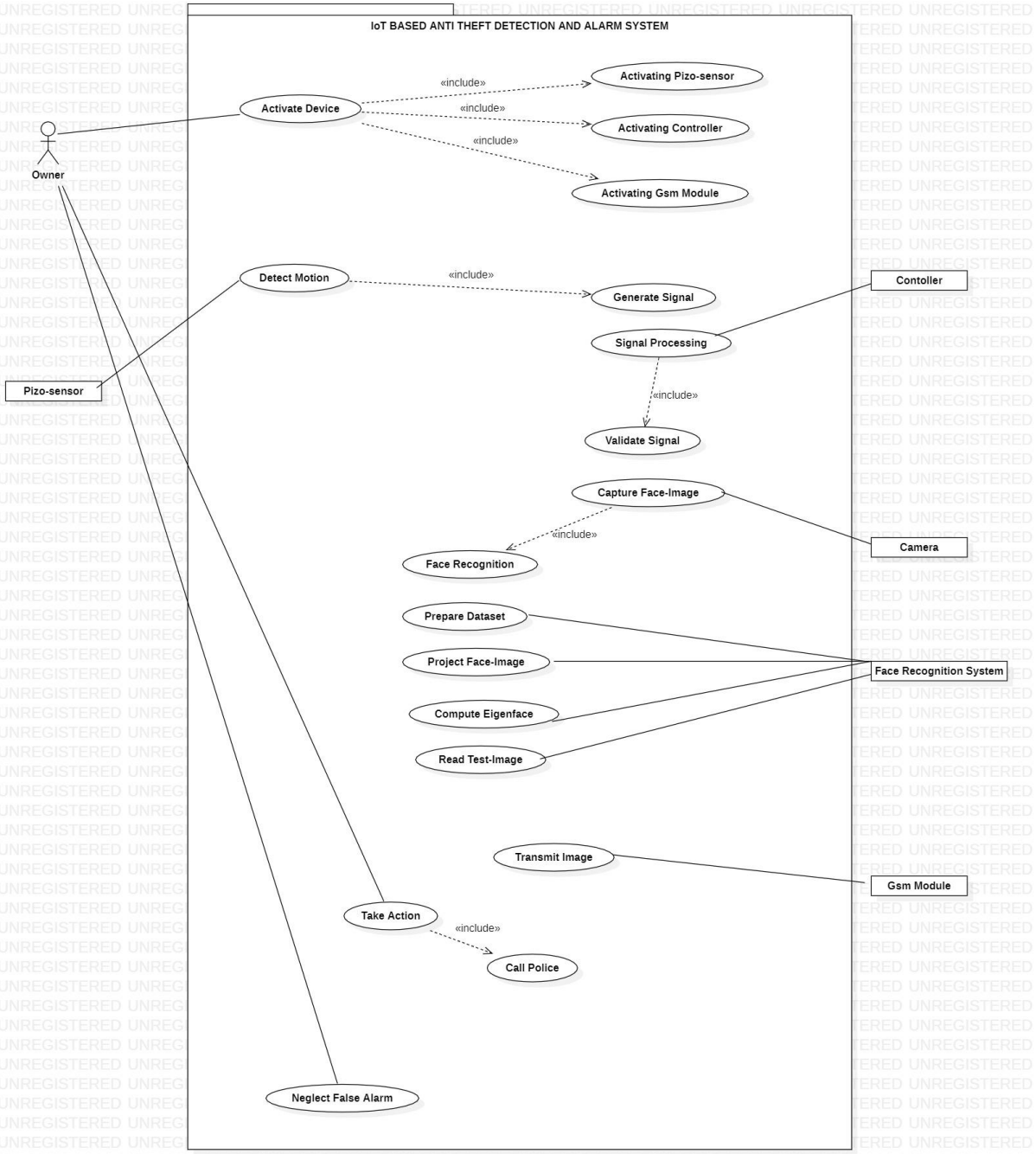
6.Scope:

- This project involves the technology that helps in detecting the floor movement of the thief using a piezo-sensor.
- It alerts the user through mail after undergoing the processes like capturing face image and face-recognition technique.

6.Process Flow:



7.Use Case Diagram:



Actors Goals:

Homeowner:

- The main actor around which the whole process revolves.
- This actor is the one who initiates the process by activating all devices like piezo-sensor, Gsm-module, raspberry-pi controller.
- On receiving the images and alert message the homeowner should verify if the thief is present.
- If so then he must inform the police station and neighborhood for rescue.

Piezo-sensor:

- This is also a primary actor where the actual process begins.
- This sensor will detect the motion once the thief steps onto the floor-mat.
- On detecting the motion, it checks the minimum standing time of that person.
- If the time is greater than the minimum time then it transmits the input signal to the raspberry-pi controller for validating the signal.

Raspberry-pi controller:

- This is the secondary actor which on validating the signal it turns on the camera.

Camera:

- ✓ This is the secondary actor which will capture the images of the thief and initiates the face-recognizer.
- ✓ On receiving the notification from the face-recognizer it will transmit the images to the Gsm-module.

Face-Recognizer:

- ❖ This is the secondary actor which on receiving the images from the camera should undergo a face-recognition technique using eigen-face algorithm.
- ❖ It should notify the camera whether the face is recognized or unrecognized face.

INCLUDE-USE CASE DESCRIPTIONS:

- Activating piezo-sensor is included in activate device use case. The owner should activate piezo-sensor.
- Activating controller is included in activate device use case. The owner should activate controller.
- Activating gsm-module is included in activate device use case. The owner should activate Gsm-module.
- Generate signal is included in detect motion use case. The piezo-sensor will generate an input signal once a motion is detected.
- Validate signal is included in signal processing use case. The controller must compulsorily validate the signal for signal processing.
- Face-recognition is included in capture face image use case. The camera must undergo face-recognition technique to find if the face is known or unknown.
- Call police is included in take action use case. The owner should take action if the thief is real by calling the police.

USE CASE DESCRIPTIONS:

Activate Device:

Level: User level.

Actors: Home-Owner.

Stakeholders and Interests:

Home-owner: The Home-Owner should activate the device to detect the entry of thief.

Pre-Condition: The Home-Owner should have installed the device.

Post-Condition: After activating the device 'Detect Motion' use case would be carried out.

Detect Motion:

Level: System level.

Actors: Piezo-sensor.

Stakeholders and Interests:

Piezo-Sensor: When a thief enters in the house and steps on the floor, it is immediately sensed by this sensor.

Pre-Condition: The house owner should have been activated the IoT based anti-theft detection and alarm system.

Post-Condition: Once the motion is detected, the input signal would be generated for processing it.

Signal Processing:

Level: System level.

Actors: Controller.

Stakeholders and Interests:

Controller: The input signal must be validated.

Pre-Condition: The motion must be detected and the signal should be passed to the controller.

Post-Condition: Once the signal is validated, camera should be turned on to capture images.

Capture Face-image:

Level: System level.

Actors: Camera

Stakeholders and Interests:

Camera: The Controller moves the camera to the area where the movement was detected to capture face image.

Pre-Condition: The camera should be turned on.

Post-Condition: The face-images would be captured and store in the system.

Prepare Dataset:

Level: User level.

Actors: Face recognition system and home owner.

Stakeholders and Interests:

Face recognition system and Home-owner: The training set of images must be obtained for face recognition process.

Pre-Condition: The camera should capture face image in order to process for face recognition process and be connected to the face recognition system.

Post-Condition: All the train set images will be stored in the database in order to avoid false alarms.

Project Face-image:

Level: System level.

Actors: Face recognition system.

Stakeholders and Interests:

Face recognition system: It must compute the mean image of the training set and find deviation.

Pre-Condition: Training set images must be prepared and transformed to vector.

Post-Condition: On projecting face image, 'compute eigenfaces' use case would be carried out.

Compute Eigen-faces:

Level: System level.

Actors: Face recognition system.

Stakeholders and Interests:

Face recognition system: The system will compute eigen faces.

Pre-Condition: The average face image must be computed.

Post-Condition: The eigen vectors would be computed.

Work flow:

- ✓ Find covariance matrix.
- ✓ The eigen values are found for the covariance matrix and hence eigen vectors are computed.

Read Test-image:

Level: System level.

Actors: Face recognition system.

Stakeholders and Interests:

Face recognition system: The system should read the test image captured by the camera.

Pre-Condition: The camera should capture the face image.

Post-Condition: The system would be able to recognise if the face is recognized or unrecognized.

Workflow:

- ✓ Compute feature vector: Test image is transformed to eigenface components.
- ✓ Compute Euclidean distance: Distance between eigenface of test image and the previously computed eigenfaces are evaluated.

Transmit Image:

Level: User level.

Actors: GSM module.

Stakeholders and Interests:

GSM module: The captured image will be transmitted to the home owner with an alert sound.

Pre-Condition: The owner should always be connected to the system and should have good internet connection.

Post-Condition: The owner should check if the thief is real or not.

Take Action:

Level: User level.

Actors: Home owner.

Stakeholders and Interests:

Home-owner: The home owner should take action by calling the police if the thief is real.

Pre-Condition: The home owner should have received the images taken by the camera.

Post-Condition: The thefts which were about to happen would be stopped and the thief would be caught.

Neglect False Alarm:

Level: User level.

Actors: Home owner.

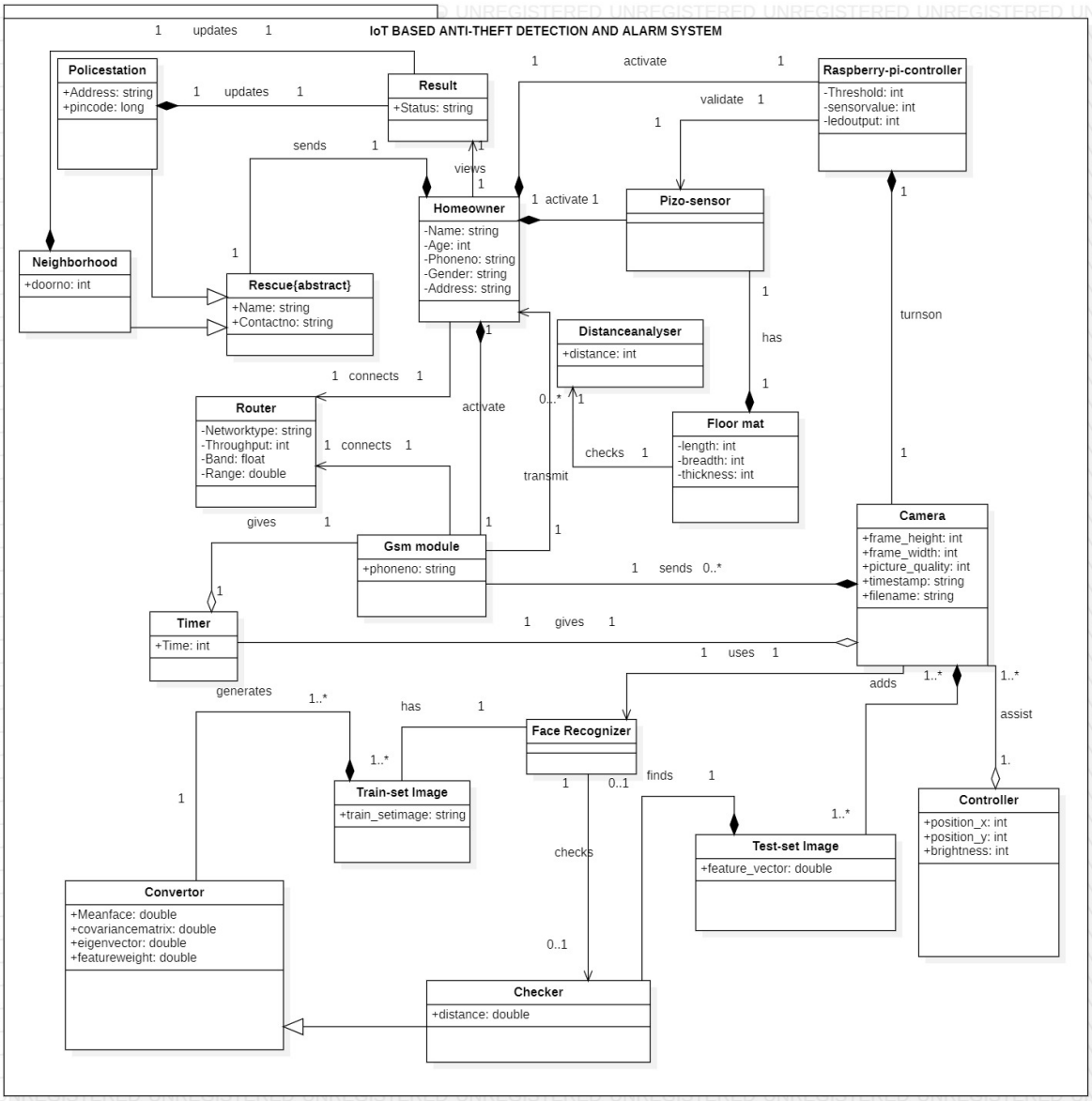
Stakeholders and Interests:

Home-owner: Home owner can abort the alarm if it happened to be a false alarm.

Pre-Condition: The home owner should have received the images taken by the camera.

Post-Condition: The false alarm will not be executed.

8.DOMAIN MODEL:



CLASSES:

- Homeowner
- Pizo-sensor
- Floor mat
- Distanceanalyser
- Rashberry-pi-controller
- Camera
- Controller
- Face Recognizer
- Train-set Image
- Test-set Image

- Convertor
- Checker
- Router
- Gsm module
- Timer
- Result
- Rescue(abstract)
  i)Neighbourhood
  ii)Policestation

ATTRIBUTES:

Homeowner:

1.Name

2.Age

3.Phoneno

4.Gender

5.Address

Floor mat:

1.length

2.breadth

3.thickness

Distanceanalyser:

1.distance

Rashberry-pi-controller:

1.Threshold

2.sensorvalue

3.ledoutput

Camera:

1.frame_height

2.frame_width

3.picture_quality

4.timestamp

5.filename

Controller:

1.position_x

2.position_y

3.brightness

Train-set Image:

1.train_setimage

Test-set Image:

1.feature_vector

Convertor:

1.Meanface

2.covariancematrix

3.eigenvector

4.featureweight

Checker:

1.distance

Gsm module:

1.phoneno

Timer:

1.Time

Router:

1.Networktype

2.Throughput

3.Band

4.Range

Result:

1.Status

Rescue{abstract}:

1.Name

2.Contactno

i)Neighbourhood:

1.doorno

ii)Policestation:

1.Address

2.pincode

DATA DICTIONARY:

1) **Pizo-sensor**: Pizo-sensor is an actor which will detect the motion if a power steps on the floor mat.
2) **Raspberry-pi-controller**: It is an actor which will validate the signal generated by the pizo-sensor.
3) **Camera**: The raspberry-pi-controller should turn-on the camera if the signal processed is valid.
4) **Controller**: The control class will assist the camera by adjusting the frame height, frame width and brightness in order to capture a clear face image.
5) **Face-recognizer**: Face-recognizer is a class which is used by the camera to undergo face-recognition technique to find if the person is a known person or unknown person.
6) **Train-set image**: Train-set image is a class which stores the list of training images for the face recognition process.
7) **Convertor**: This part of the class performs the main computation part of the eigen face-algorithm. Thus, computation mainly includes finding the mean image, computing matrix, computing eigen vector of trainset image and the feature-weight of train-set image.
8) **Test-set image:** The camera would add the captured images as input image to the test-set image class. The test set image class consists of the mean face and feature-vector of the test image.
9) **Checker**: This class is used by the test-set image to find the Euclidean distance. The face-recognition also uses this class to check if the face is recognized or unrecognized.
10) **Homeowner:** This is the main actor of this system. The homeowner should activate the pizo-sensor, raspberry-pi-controller and Gsm module for a smooth moving process.
11) **Gsm-module**: This is an actor which will transmit the images to the home owner which are captured by the camera.
12) **Timer**: This class is used by the Gsm-module to check the time of the captured image.
13) **Rescue**: This class is used by the homeowner to inform about the entry of thief.
14) **Police station**: This is the subclass of rescue to get the details of the homeowner including address, pincode and to take actions and then updates the status.

15) **Neighbourhood**: This is the subclass of rescue to get the details of the homeowner which includes doorno and to take actions and then updates the status.
16) **Floormat:** This class contains the details like length, breadth, thickness. It is used by the pizo-sensor class to place the sensor under the floormat
17) **Status:** The homeowner can check out the status whether the police have arrived yet or not.
18) **Router:** This class is necessary for routine the data packets from source to destination.
19) **Distance analyser:** This class will be used by the floor mat to check the minimum distance to place the mat.

ASSOCIATION AND MULTIPLICITY:

- Home owner connects to router
  Multiplicity: One home owner should connect to one router. (1-1)
- Home owner views status
  Multiplicity: One home owner will view the statusof police arrival. (1-1)
- Raspberry pi controller validates the signal of pizo-sensor
  Multiplicity: One raspberry pi controller should validate the signal generated by the sensor. (1-1)
- Camera uses face recognition
  Multiplicity: One camera uses face recognition class to detect faces. (1-1)
- Face recognizer has train set images
  Multiplicity: One face recognizer has many train set images. (1-1..*)
- Face recognizer checks checker
  Multiplicity: One face recognizer uses the checker class. (1-0..1)

- Gsm module uses camera
  Multiplicity: One gsm module uses zero or more images taken by camera. (1-0..*)
- Gsm module transmit to homeowner
  Multiplicity: One gsm module will transmit many images or zero images. (1-0..*)
- Gsm module connects to router
  Multiplicity: One gsm module connects to one router. (1-1)
- Floor mat checks distance analyser
  Multiplicity: One floor mat will check the distance.
  (1-1)

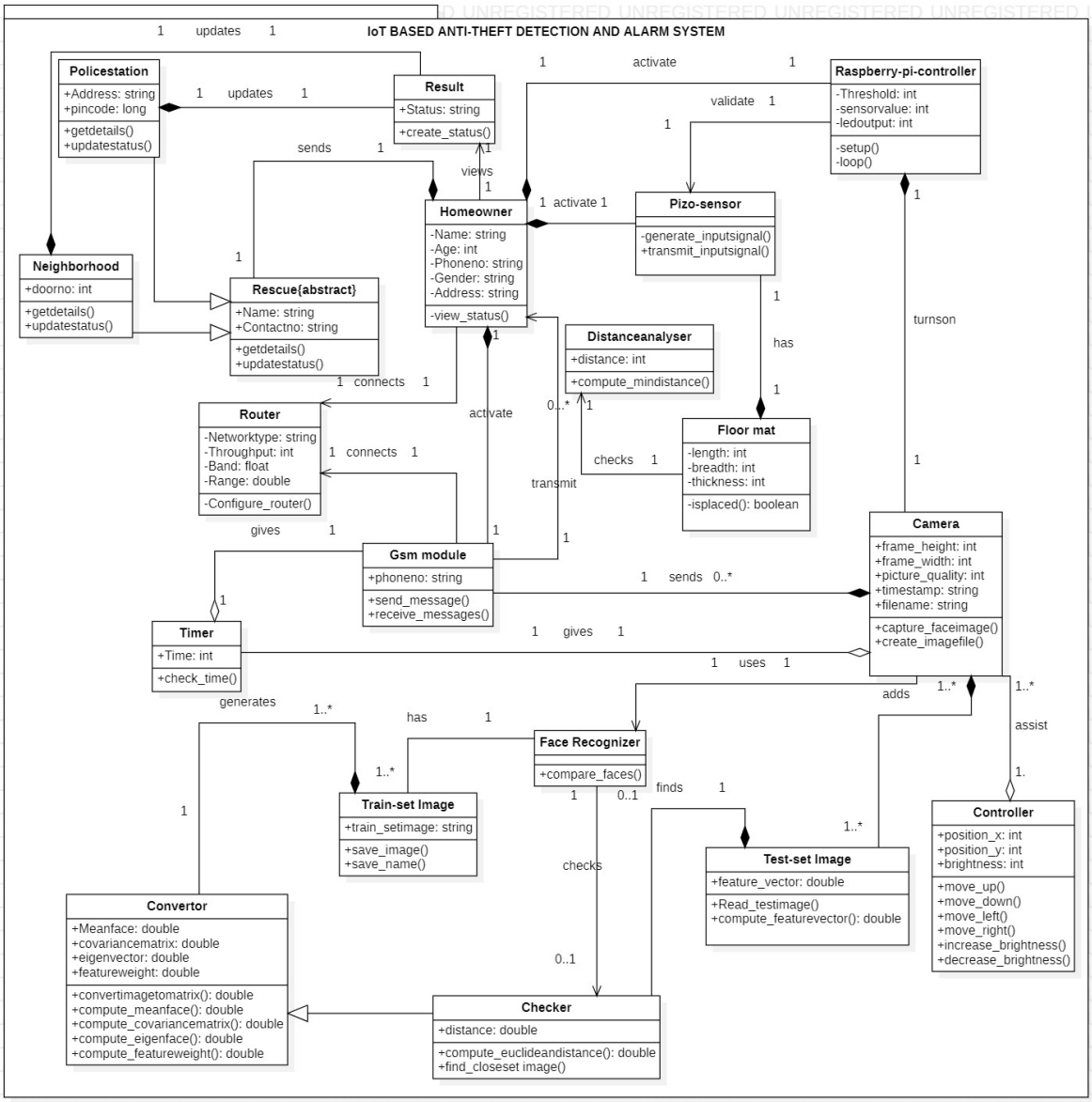
AGGREGATION:

- Timer is gives GSMmodule
  Multiplicity: One timer is checked by gsm module checked for duration.(1-1)
- Camera gives Timer
  Multiplicity: One Camera helps the timer to check the duration. (1-1)
- Controller assists the camera
  Multiplicity: One control class will assist more images taken by camera. (1-1. .*)

COMPOSITION:

- Homeowner activates pizo-sensor
  Multiplicity: One homeowner should activate one pizo-sensor. (1-1)
- Homeowner activates Raspberry pi controller
  Multiplicity: One homeowner should activate one raspberry pi controller. (1-1)
- Home owner activates Gsm module
  Multiplicity: One home owner should activate one gsm module. (1-1)
- Home owner informs rescue
  Multiplicity: One homeowner should inform the police station about the thief entry. (1-1)
- Floormat has Pizo-sensor
  Multiplicity: One floormat has one pizo-sensor. (1-1)
- Raspberry pi controller turns on camera
  Multiplicity: One raspberry pi controller must turn on the camera to capture the images. (1-1)
- Camera adds Test set images
  Multiplicity: Many images taken by the camera will be added to the train set image. (1. .* - 1. .*)
- Trainset image generates the converter
  Multiplicity: Many train set images will be converted into eigen vectors. (1..*-1)
- Police station update the status
  Multiplicity: One police station can update the status. (1-1)
- Neighbourhood update the status
  Multiplicity: One police neighbourhood can update the status. (1-1)
- Testsetimage finds distance of checker
  Multiplicity: One testsetimage will check the computed Euclidean distance of checker. (1-0..1).

9.CLASS DIAGRAM:



CLASSES:

- Homeowner
- Pizo-sensor
- Floor mat
- Distanceanalyser
- Rashberry-pi-controller
- Camera
- Controller
- Face Recognizer
- Train-set Image
- Test-set Image

- Convertor
- Checker
- Router
- Gsm module
- Timer
- Result
- Rescue(abstract)
  i)Neighbourhood
  ii)Policestation

ATTRIBUTES:

Homeowner:

1.Name

2.Age

3.Phoneno

4.Gender

5.Address

Floor mat:

1.length

2.breadth

3.thickness

Distanceanalyser:

1.distance

Rashberry-pi-controller:

1.Threshold

2.sensorvalue

3.ledoutput

Camera:

1.frame_height

2.frame_width

3.picture_quality

4.timestamp

5.filename

Controller:

1.position_x

2.position_y

3.brightness

Train-set Image:

1.train_setimage

Test-set Image:

1.feature_vector

Convertor:

1.Meanface

2.covariancematrix

3.eigenvector

4.featureweight

Checker:

1.distance

Gsm module:

1.phoneno

Timer:

1.Time

Router:

1.Networktype

2.Throughput

3.Band

4.Range

Result:

1.Status

Rescue{abstract}:

1.Name

2.Contactno

i)Neighbourhood:

1.doorno

ii)Policestation:

1.Address

2.pincode

OPERATIONS:

Homeowner:

1.view_status: The status from the result class can be viewed.

Pizo-sensor:

1.generate_inputsignal(): The input signal is generated by the pizo-sensor

2.transmit_inputsignal(): The generated input signal is transmitted to Rashberry-pi-controller

Floor mat:

1.isplaced(): It is used to check whether the pizo-sensor is placed under the floormat.

Distanceanalyser:

1.compute_mindistance(): It is used to compute minimum distance of the floormat from the door.

Rashberry-pi-controller:

1.setup(): To declare led connected pin to output.

2.loop(): To read analog voltage from sensor.

Camera:

1.capture_faceimage(): It is used to capture the face-image.

2.create_imagefile(): It is used to create the image file.

Controller:

1.move_up(): It moves tha camera upwards.

2.move_down(): It moves the camera downwards.

3.move_left(): It moves the camera leftwards.

4.move_right(): It moves the camera rightwards.

5.increase_brightness(): It increases the brightness of the captured image.

6.decrease_brightness(): It decreases the brightness of the captured image.

Face Recognizer:

1.compare_faces(): It is used to compare the faces of the captured image with the train set images.

Train-set image:

1.save_image(): It is used to save the image.

2.save_name(): It is used to save the name of the train-set image.

Test-set image:

1.Read_testimage(): It is used to read the newly captured image.

2.compute_featurevector(): It is used to compute the feature vector the test set image.

Convertor:

1.convertimagetomatrix(): It converts the image to matrix.

2.compute_meanface(): It computes the mean face of the generated matrix.

3.compute_covariancematrix(): It computes the covariance matrix.

4.compute_eigenface(): It computes the eigenface.

5.compute_featureweight(): It computes the feature weight of the train set images.

Checker:

1.compute_euclideandistance(): It will compute the Euclidean distance.

2.find_closestimage(): It will find the closest image.

Gsm module:

1.send_message(): It will send message to the homeowner.

2.receive_message(): It will receive images from the camera.

Timer:

1.check_time(): It is used to check time of the captured image from the camera.

Router:

1.Configure_router():It is used to configure the router.

Rescue{abstract}:

1.getdetails():It is used to get details of the homeowner.

2.update_status(): It is used to update the status if the rescue operation have been carried out or not.

Result:

1.create_status(): It is used to create the status.

DATA DICTIONARY:

1) **Pizo-sensor**: Pizo-sensor is an actor which will detect the motion if a power steps on the floor mat.
2) **Raspberry-pi-controller**: It is an actor which will validate the signal generated by the pizo-sensor.
3) **Camera**: The raspberry-pi-controller should turn-on the camera if the signal processed is valid.
   Controller: The control class will assist the camera by adjusting the frame height, frame width and brightness in order to capture a clear face image.
4) **Face-recognizer**: Face-recognizer is a class which is used by the camera to undergo face-recognition technique to find if the person is a known person or unknown person.
5) **Train-set image**: Train-set image is a class which stores the list of training images for the face recognition process.
6) **Convertor**: This part of the class performs the main computation part of the eigen face-algorithm. Thus, computation mainly includes finding the mean image, computing matrix, computing eigen vector of trainset image and the feature-weight of train-set image.
7) **Test-set image:** The camera would add the captured images as input image to the test-set image class. The test set image class consists of the mean face and feature-vector of the test image.
8) **Checker**: This class is used by the test-set image to find the Euclidean distance. The face-recognition also uses this class to check if the face is recognized or unrecognized.
9) **Homeowner:** This is the main actor of this system. The homeowner should activate the pizo-sensor, raspberry-pi-controller and Gsm module for a smooth moving process.
10) **Gsm-module**: This is an actor which will transmit the images to the home owner which are captured by the camera.
11) **Timer**: This class is used by the Gsm-module to check the time of the captured image.
12) **Rescue**: This class is used by the homeowner to inform about the entry of thief.

13) **Police station**: This is the subclass of rescue to get the details of the homeowner including address, pincode and to take actions and then updates the status.
14) **Neighbourhood**: This is the subclass of rescue to get the details of the homeowner which includes doorno and to take actions and then updates the status.
15) **Floormat:** This class contains the details like length, breadth, thickness. It is used by the pizo-sensor class to place the sensor under the floormat
16) **Status:** The homeowner can check out the status whether the police have arrived yet or not.
17) **Router:** This class is necessary for routine the data packets from source to destination.
18) **Distance analyser:** This class will be used by the floor mat to check the minimum distance to place the mat.

ASSOCIATION AND MULTIPLICITY:

- Home owner connects to router
  Multiplicity: One home owner should connect to one router. (1-1)
- Home owner views status
  Multiplicity: One home owner will view the statusof police arrival. (1-1)
- Raspberry pi controller validates the signal of pizo-sensor
  Multiplicity: One raspberry pi controller should validate the signal generated by the sensor. (1-1)
- Camera uses face recognition
  Multiplicity: One camera uses face recognition class to detect faces. (1-1)
- Face recognizer has train set images
  Multiplicity: One face recognizer has many train set images. (1-1..*)
- Face recognizer checks checker
  Multiplicity: One face recognizer uses the checker class. (1-0..1)
- Gsm module uses camera
  Multiplicity: One gsm module uses zero or more images taken by camera. (1-0..*)
- Gsm module transmit to homeowner
  Multiplicity: One gsm module will transmit many images or zero images. (1-0..*)
- Gsm module connects to router
  Multiplicity: One gsm module connects to one router. (1-1)
- Floor mat checks distance analyser
  Multiplicity: One floor mat will check the distance.
  (1-1)

AGGREGATION:

- Timer is gives GSMmodule
  Multiplicity: One timer is checked by gsm module checked for duration.(1-1)
- Camera gives Timer
  Multiplicity: One Camera helps the timer to check the duration. (1-1)
- Controller assists the camera
  Multiplicity: One control class will assist more images taken by camera. (1-1. .*)

COMPOSITION:

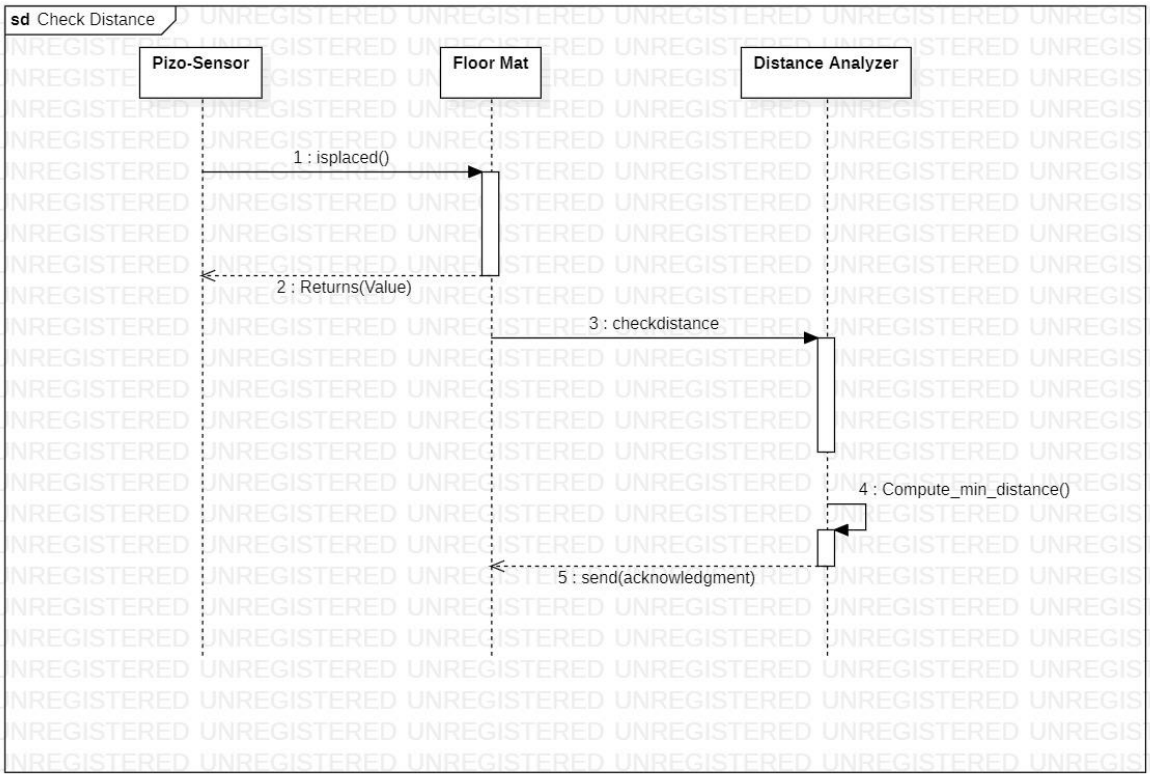
- Homeowner activates pizo-sensor
  Multiplicity: One homeowner should activate one pizo-sensor. (1-1)
- Homeowner activates Raspberry pi controller
  Multiplicity: One homeowner should activate one raspberry pi controller. (1-1)
- Home owner activates Gsm module
  Multiplicity: One home owner should activate one gsm module. (1-1)
- Home owner informs rescue
  Multiplicity: One homeowner should inform the police station about the thief entry. (1-1)
- Floormat has Pizo-sensor
  Multiplicity: One floormat has one pizo-sensor. (1-1)
- Raspberry pi controller turns on camera
  Multiplicity: One raspberry pi controller must turn on the camera to capture the images. (1-1)
- Camera adds Test set images
  Multiplicity: Many images taken by the camera will be added to the train set image. (1. .* - 1. .*)
- Trainset image generates the converter
  Multiplicity: Many train set images will be converted into eigen vectors. (1..*-1)
- Police station update the status
  Multiplicity: One police station can update the status. (1-1)
- Neighbourhood update the status
  Multiplicity: One police neighbourhood can update the status. (1-1)
- Testsetimage finds distance of checker
  Multiplicity: One testsetimage will check the computed Euclidean distance of checker. (1-0..1).

CRC:

CLASS	RESPONSIBILITIES	COLLOBORATIONS
Homeowner	• view_status()	• Pizosensor • Rashberry_pi_controller • GSMModule • Policestation • Router • Status
Floormat	• isplaced():boolean	• Distance_analyser
Rashberry_pi_controller	• setup() • loop()	• Pizosensor • Camera
Pizosensor	• Generate_inputsignal() • Transmit_inputsignal()	• Rashberry_pi_controller • Floormat
Camera	• capture_faceimage() • create_imagefile()	• FaceRecognizer • GSMModule • Test_set_image
Controller	• move_up() • move_down() • move_left() • move_right() • increase_brightness() • decrease_brightness()	• Camera
FaceRecognizer	• compare_faces()	• Train_set_image • Checker
Train_set_image	• save_image() • save_name()	• Converter
Test_set_image	• Read_testimage() • compute_featurevector():double[]	• Checker • Camera
Converter	• convertimagetomatrix():double[][] • compute_meanface():double[] • compute_covariancematrix():double[][] • compute_eigenface():double[][] • compute_featureweight():double[]	• Train_set_image
Checker	• Compute_euclideandistance():double • find_closestimage()	• Test_set_image • Train_set_image
GSMModule	• send_message() • receive_mesaage()	• Timer • Router • Homeowner
Timer	• checktime()	• Camera
Rescue	• getdetails()	• Policestation • Neighbourhood
Neighbourhood	• update_status():String • getdetails()	• Status • Homeowner
Policestation	• update_status():String • getdetails()	• Status • Homeowner
status	• create_status():String	• Policestation

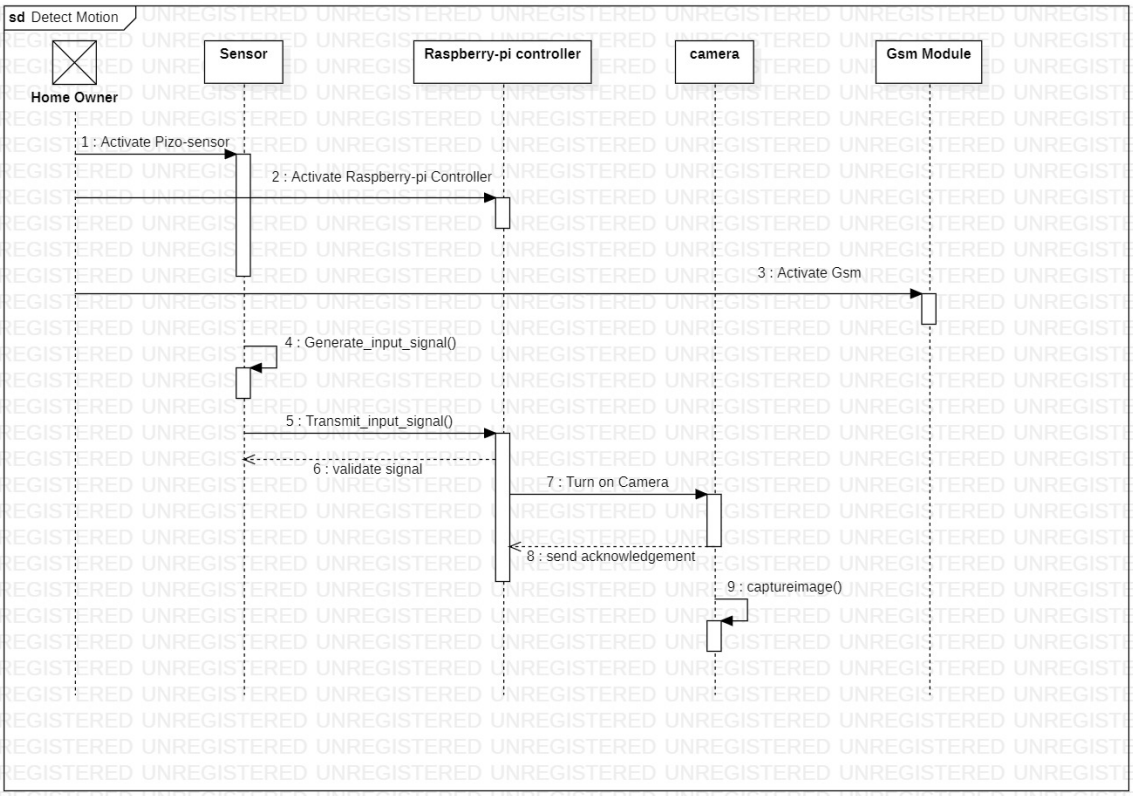
10.SEQUENCE DIAGRAM:

CHECKS-DISTANCE:



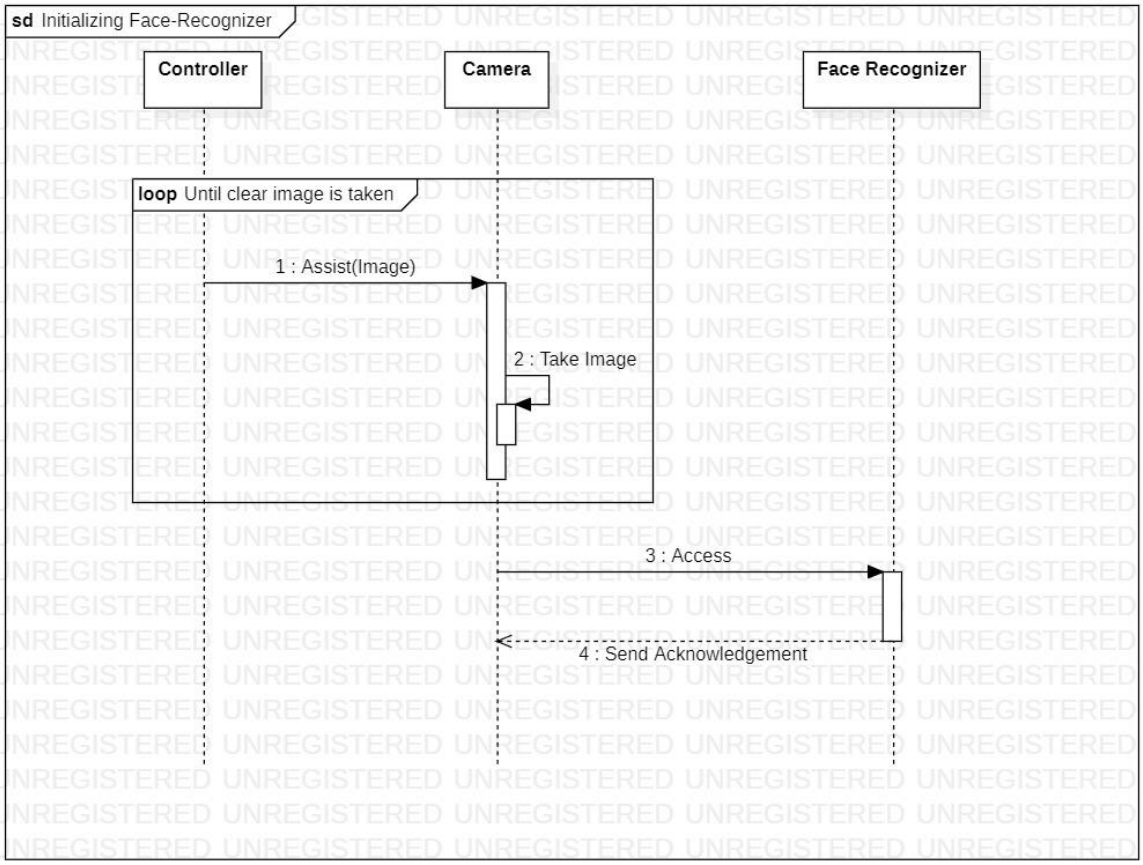
- ✓ The pizo-sensor should check whether it isplaced under the floormat using **isplaced()** method.
- ✓ In response the floormat **returns(value)** to the pizo-sensor.
- ✓ The floormat will now **check distance** with the distance-analyser.
- ✓ The distance-analyzer should now **compute_mindistance().**
- ✓ It should then **send(acknowledgment)** to the floormat.

DETECTS MOTION:



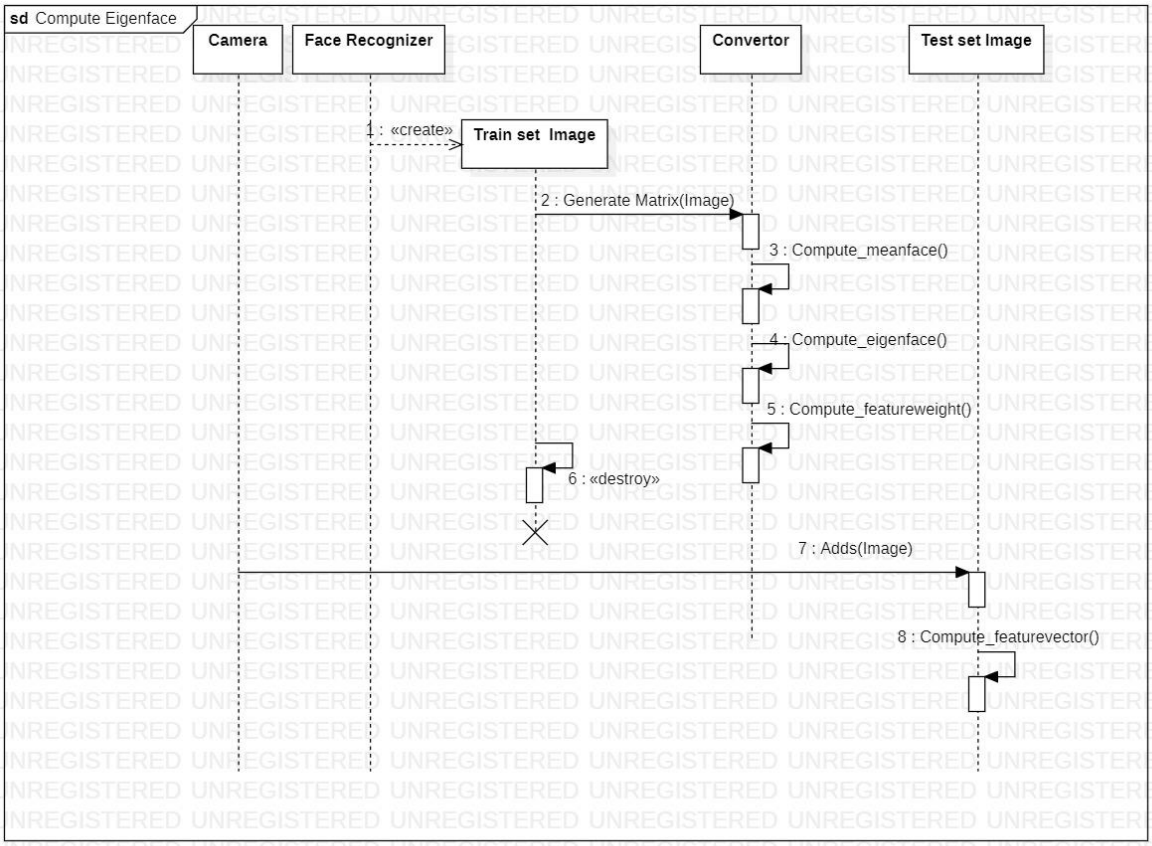
- The homeowner should **activate** the **pize-sensor, rashberry-pi controller** and **gsm-module** respectively.
- The pizo-sensor will generate an input signal on detecting motion using **Generate_input_signal()** method**.**
- Once the signal is generated it is transmitted to the rashberry-pi controller using **Transmit_input_signal()** method.
- In response the rashberry-pi controller **validates signal** generated by the pizo-sensor.
- Once the signal is valid the rashberry-pi controller **Turn on camera**.
- In response the camera will **send acknowledgement** to the rashberry-pi controller.
- The camera should capture images of the thief using **captureimage()** method.

INITIALIZING FACERECOGNIZER:



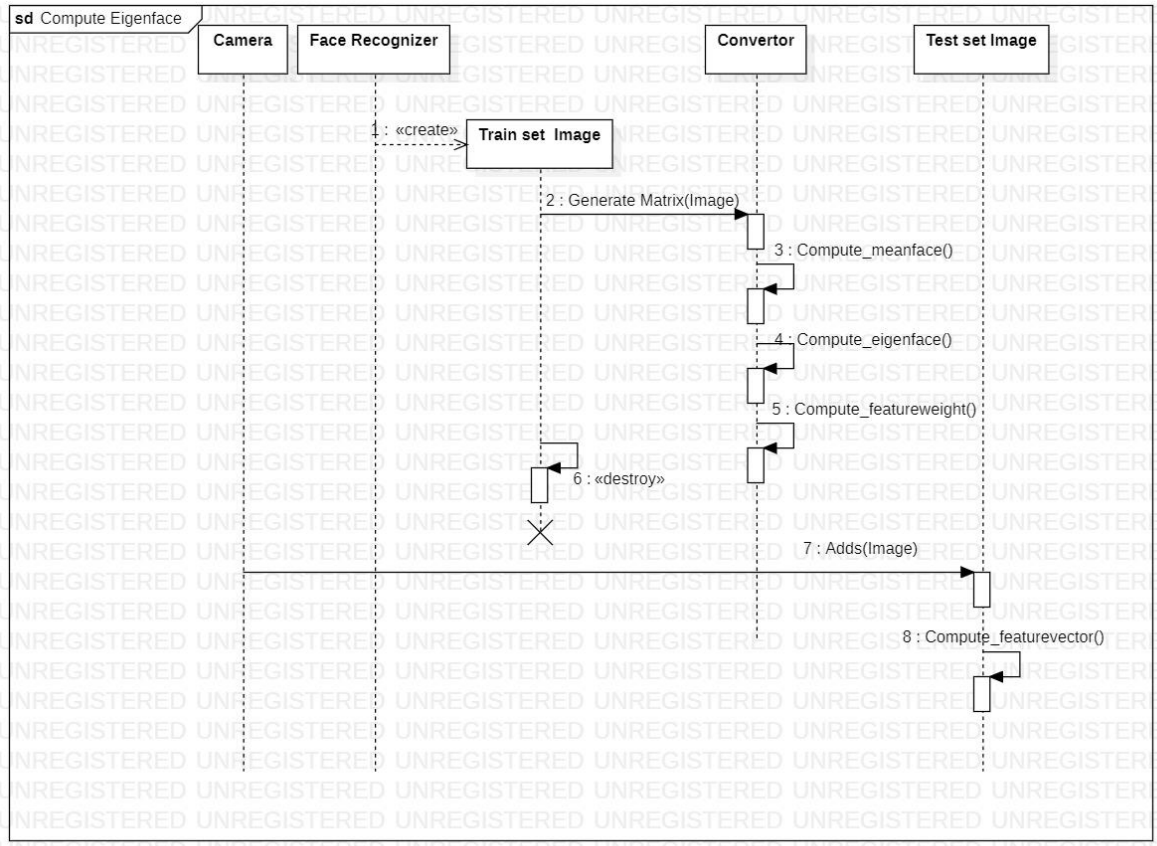
- The controller will be assisting the images taken by the camera using **Assist(image)** method.
- The camera will have to **take image**. This continues in a **loop until a clear image is taken.**
- The camera should then **Access** the face recognizer and in response the face recognizer should **send acknowledgement.**

COMPUTES EIGENFACE:



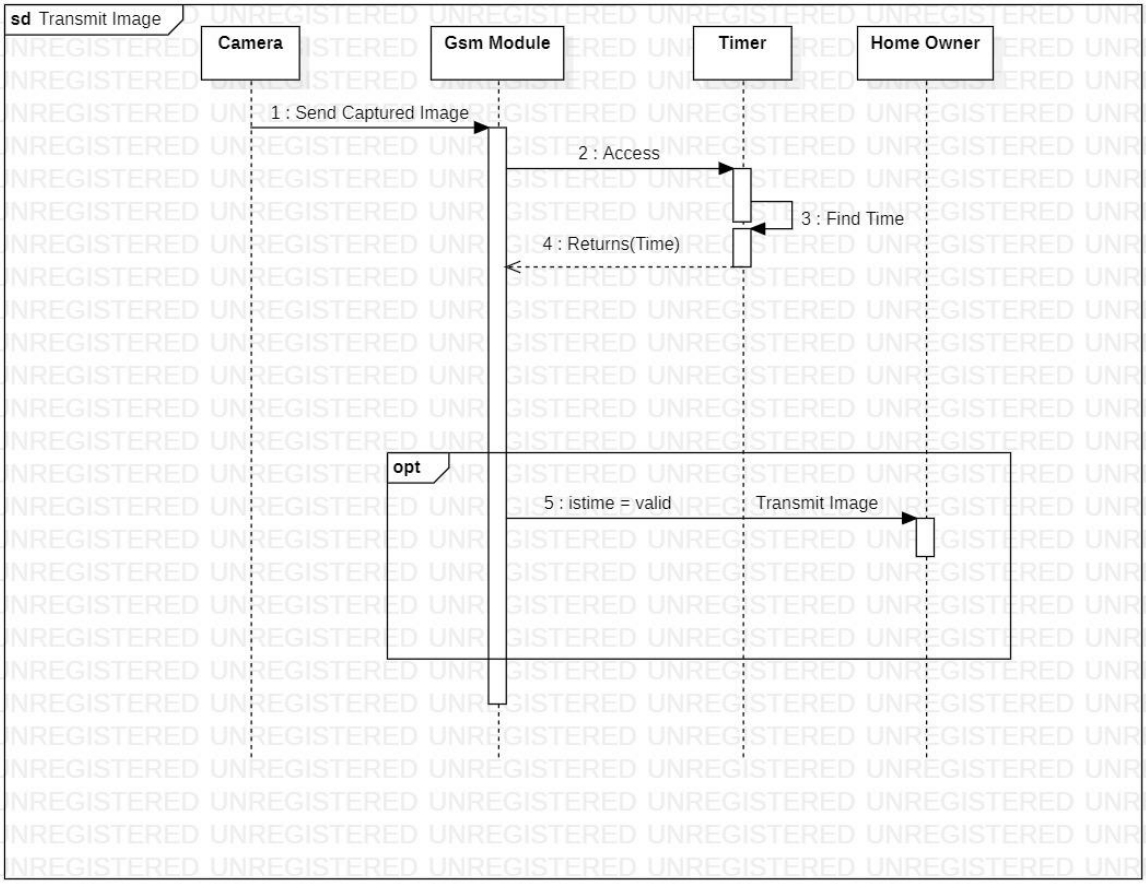
- The face recognizer will **create** an instance of **Trainsetimage.**
- The Trainset image will generate a matrix of the trainsetimages to the Convertor using the **Generate_matrix(image)** method.
- The Convertor will compute mean face of the train set images using the **compute_meanface()** method.
- It will then compute the eigenface using the meanface by the method **Compute_eigenface()** method.
- After that the Convertor will compute the feature weight of the trainsetimages using **Compute_featureweight(eigenface)** method.
- The **Trainsetimage** is now **destroyed.**
- The camera then adds the images to the Testsetimage using **Adds(image)** method.
- The Testsetimage will then compute the feature vector the added test-set image using the **compute_featurevector()** method.

FACE RECOGNITION:



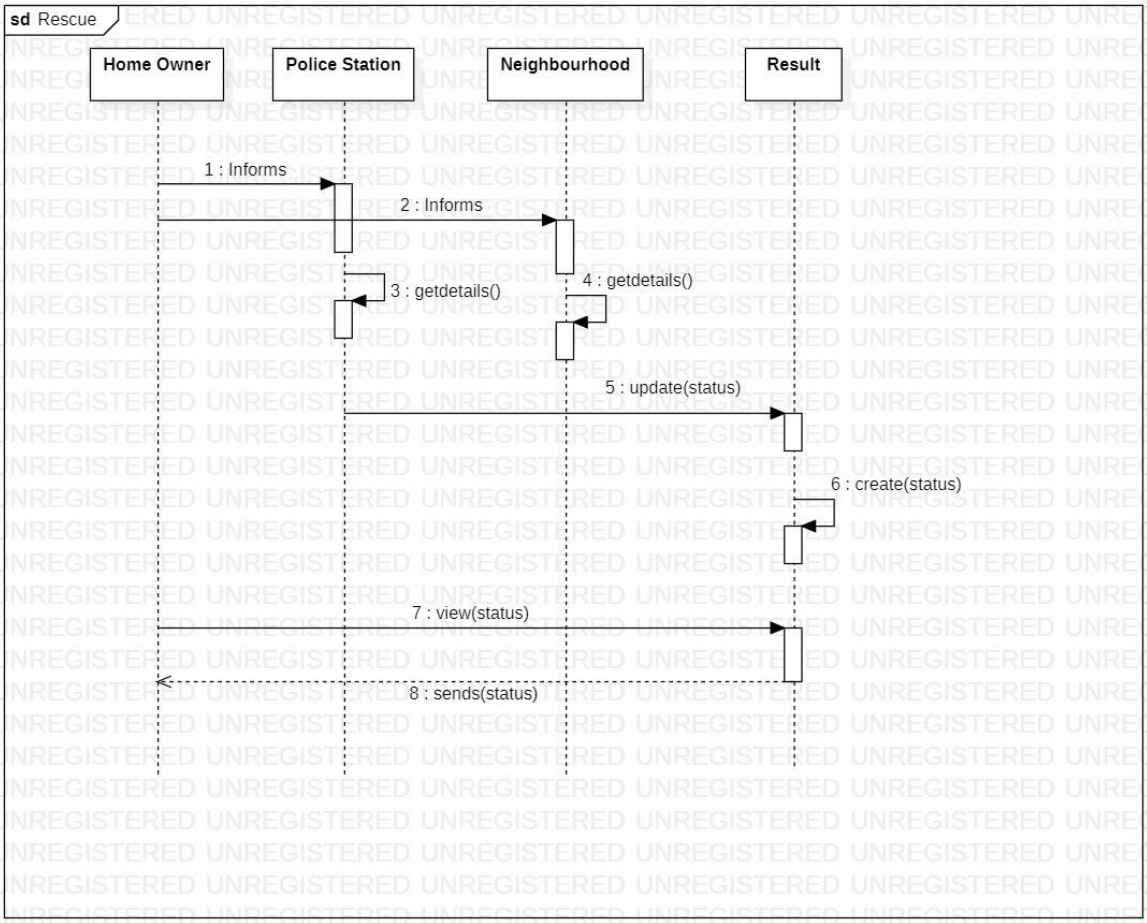
- The Testsetimage will transfer the feature vector to the checker using the **Transfer(feature_vector)** method.
- The checker should compute the Euclidean distance using the **compute_Euclideandistance()** method.
- The fragment **alt** is used. The 2 guard conditions are **"on recognition"** and **"on unrecognition".**
- Upon recognition the checker will **notify** the facerecognizer and on unrecognition the checker will read the next image from the facerecognizer using the **Readnext(image)** method.
- The Facerecognizer should **acknowledge** the **camera** whether the face is recognized or unrecognized.

TRANSMIT IMAGE:



- The camera will **sends captured image** to the gsm-module.
- The gsm module will then **access** the Timer.
- The Timer will **find time** and returns the time to the gsm-module using **Returns(time)** method.
- The fragment **opt** is used.
- The guard condition used is **"istime=valid"**.
- If the time is valid then the gsmmodule will transmit the image to homeowner using **transmit(image)** method.

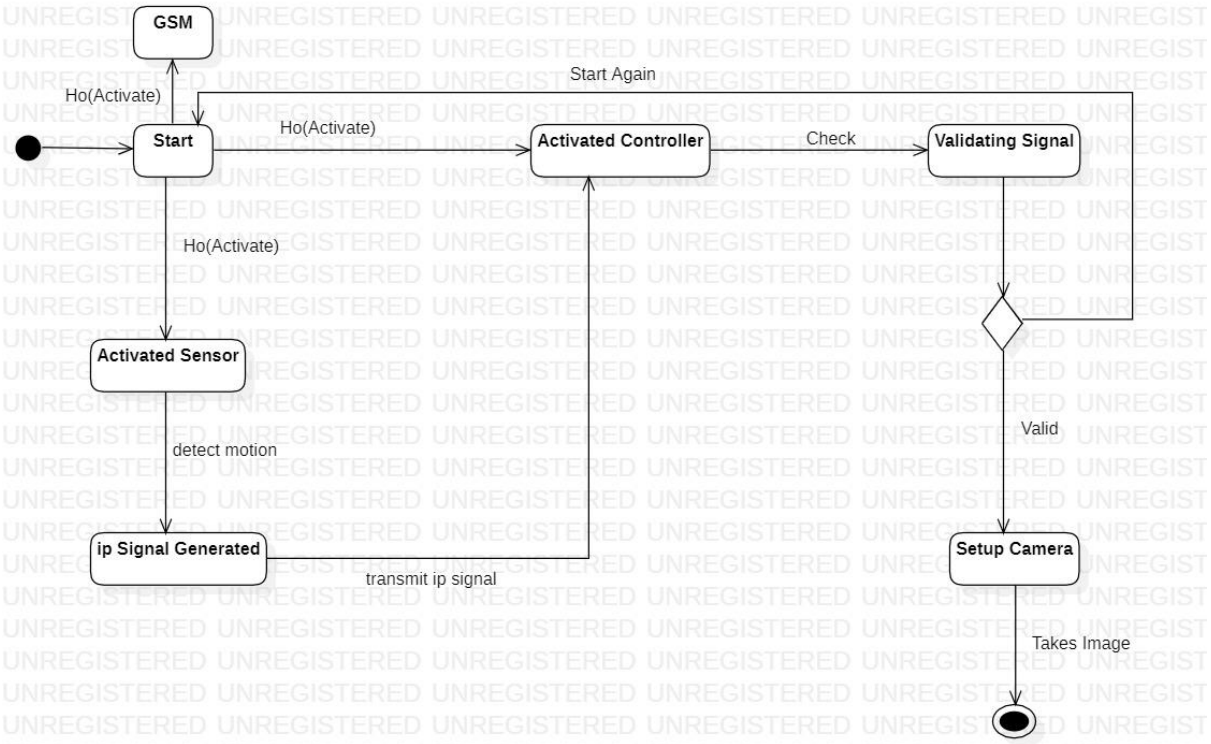
RESCUE:



- ✓ The homeowner should **inform** the policestation and neighbourhood about the entry of thief.
- ✓ The policestation must get the details of the homeowner using the **getdetails()** method.
- ✓ The Neighbourhood must also get the details of the homeowner using the **getdetails()** method.
- ✓ The policestation should update the status whether they have arrived the spot and caught the thief using the **update(status)** method.
- ✓ The Result should create the status using the **create(status)** method.
- ✓ The Homeowner should view the status whether the rescue teams have arrived by using the method **view(status).**
- ✓ The Result will then send the status to the homeowner using the **send(status)** method.

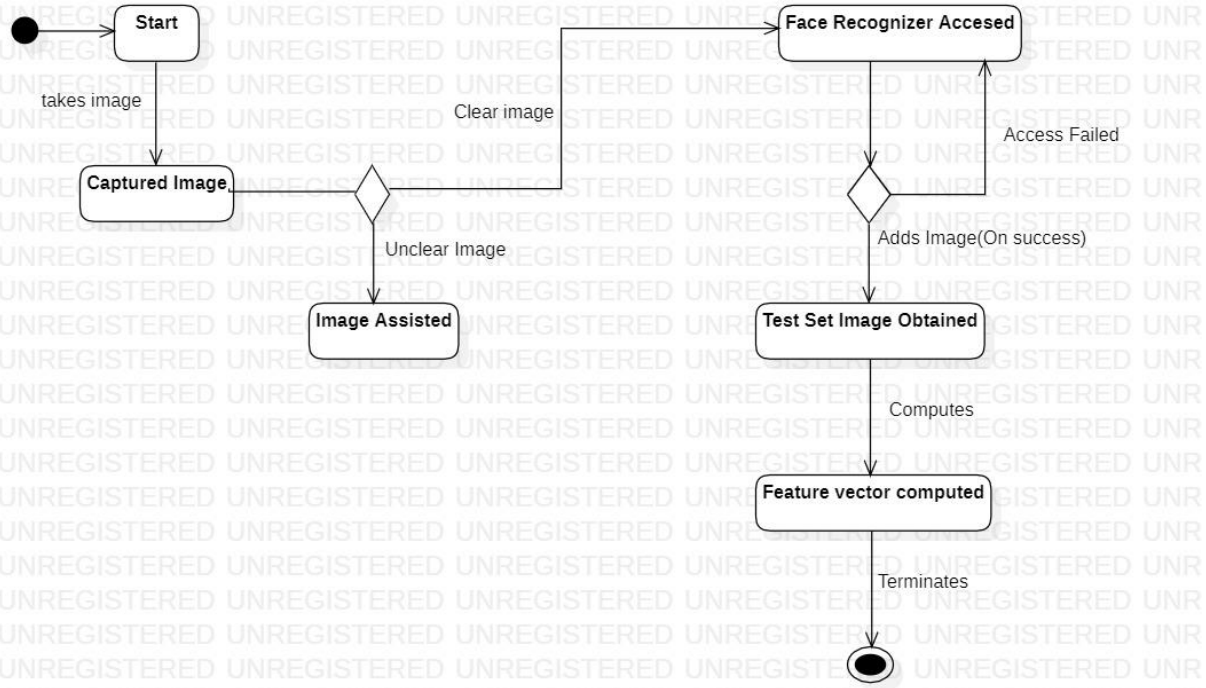
11.STATE DIAGRAM:

DETECT MOTION:



- From the **start** state, the **homeowner activates** the sensor, controller and gsm module.
- This leads to the **Activated sensor** state, **Activated controller** state and **Activated gsm-module** state respectively.
- The Activated sensor will detect the motion of the thief using the **detect motion** transition that will lead to the **input signal generator** state.
- Then there will be a **transmit input signal** transition which leads to the **Activated Controller** state.
- Now this Activated Controller state will have a **check** transition which leads to the **validating signal** state.
- Here we have a **choice** where the signal is either **valid** or **invalid.**
- **Valid** transition leads to the **setup camera** state.
- This state will have a transition **takes image.**
- This takes image transition leads to the **final** state.
- If there is an **Invalid** transition it then leads to the **start** state by having a start again transition.

INTIALIZING FACERECOGNIZER:

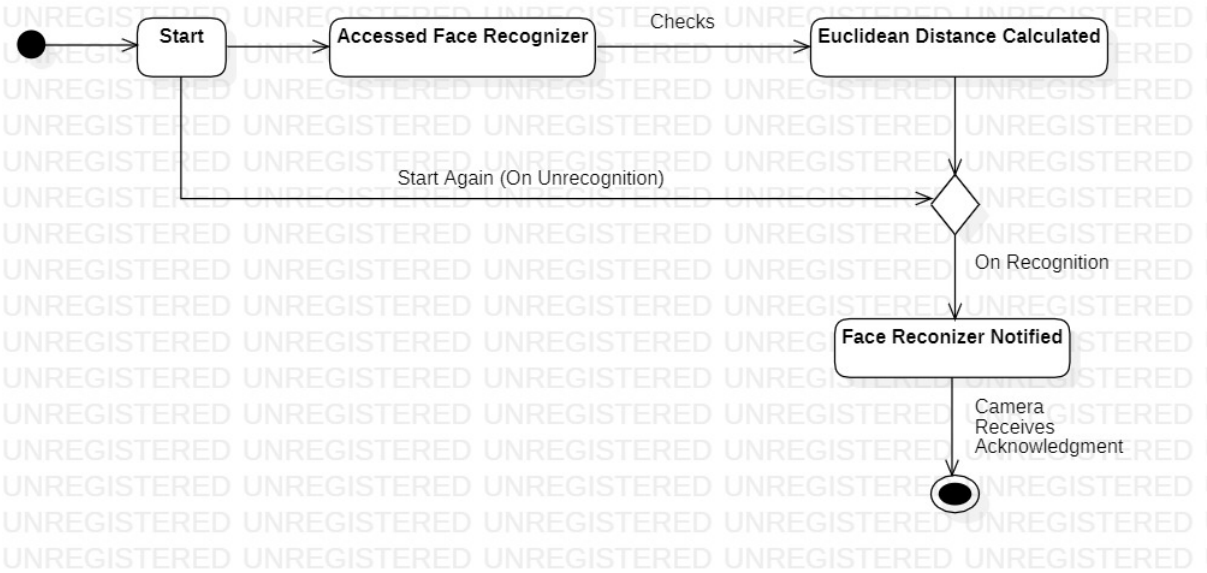


- ✓ From the **start** state, we have a transition **takes image** which leads to the **captured image** state.
- ✓ Now we have a **choice** where the image is **clear image** or **unclear image**.
- ✓ The **unclear image** transition leads to the **Image Assisted** state.
- ✓ The **clear image** transition leads to the **FaceRecognizer Accessed** state.
- ✓ We have a **choice** here where the **access failed** and **Adds image(on success).**
- ✓ The **Access Failed** transition will again lead to the **FaceRecognizer Accessed** state.
- ✓ The **Adds image(onSuccess)** transition will leads to **Testsetimage obtained** state.
- ✓ We will have to compute the feature vector of the testset image by having a transition **computes.**
- ✓ This **computes** transition leads to the **Feature vector computed** state.
- ✓ Finally the transition **terminates** leads to the **final** state.

COMPUTE EIGENFACE:

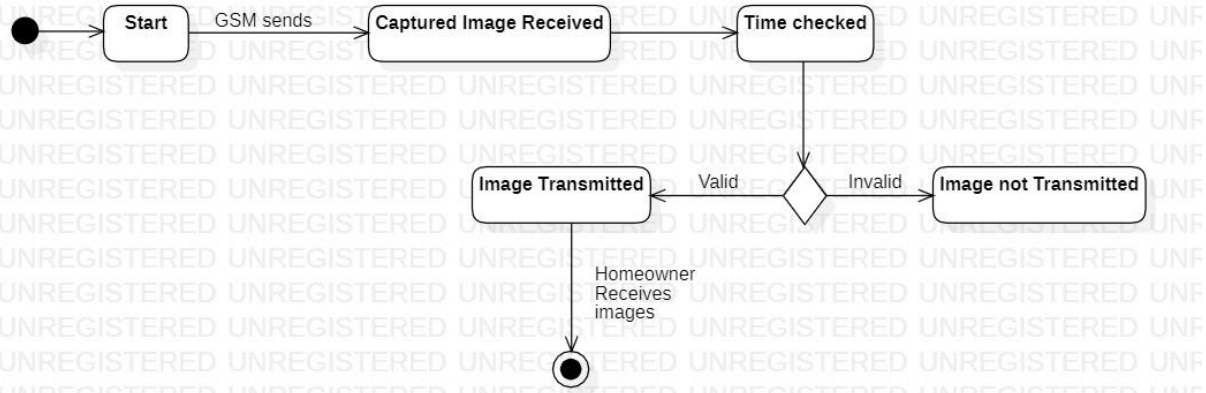
- From the initial point we have a **FaceRecognizer Accessed** state.
- This state will have a transition **creates** which leads to the **Created trainset image** state.
- The trainset image must generate the matrix for the trainset images by having a transition **generates.**
- This transition leads to the **Matrix Generated** state.
- Now we must compute the mean face of the train set images, so we will be having a transition **computes.**
- This transition leads to the **Mean face Computed** state.
- This state will have a transition **computes** to calculate the eigenface.
- This **computes** transition leads to the **Eigenface computed** state.
- This state will now enters the feature weight computed state by having a transition **computes** in order to calculate the feature weight of the trainset images.
- Now there is a final transition **terminates** from the **Feature Weight Computed** state which leads to the **final** state.

FACE RECOGNITION:



- From the **start** state we have a transition **Access** that leads to the **Accessed Face Recognizer.**
- This state will enter the **Euclidean distance calculated** state by having a transition **checks.**
- There is a **choice** where we have **On recognition** and **Start again(on unrecognition).**
- The **start again (on unrecognition)** transition leads to the **start** state.
- The transition **on recognition** leads to the **Face Recognizer Notified** state.
- Here we have a final transition **camera receives acknowledgement** that leads to the **final** state.

TRANSMIT IMAGE:



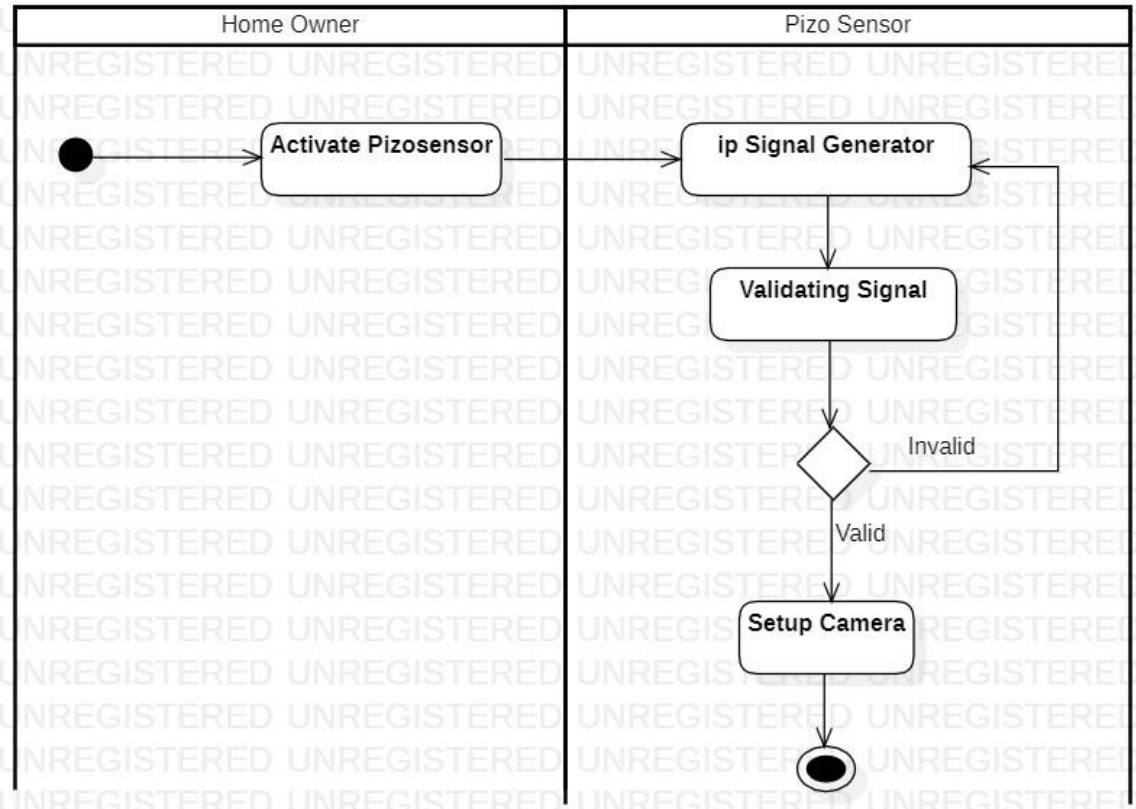
- From the **start** state the gsm send the captured image to the homeowner by having a transition **GSM sends** that leads to the **captured image received** state.
- This state should check the time by having a transition **checks** time that leads to the **Time checked** state.
- Now we have a **choice** if the time is **valid** or **invalid.**
- The time is said to be **valid** if the persons stands in the same place for more than the minimum time else it is **invalid.**
- The **valid** transition leads to the **Image transmitted** state.
- After this there is a transition **Homeowner receives image** in order to receive the image and this transition leads to the **final** state.
- The **invalid** transition leads to the **Image not transmitted** state.

RESCUE:

- ✓ From the **start** state the homeowner should inform the police station and neighborhood by having a transition **informs** that leads to the **Information Received** state.
- ✓ This state will now enter the **status updated** state once the police station and neighbourhood updates the status by having a transition **updates**.
- ✓ The status updated can be viewed by the homeowner by having a transition **views** that leads to the **status viewed** state.
- ✓ From this state we have final transition **sends** which leads to the **final** state.

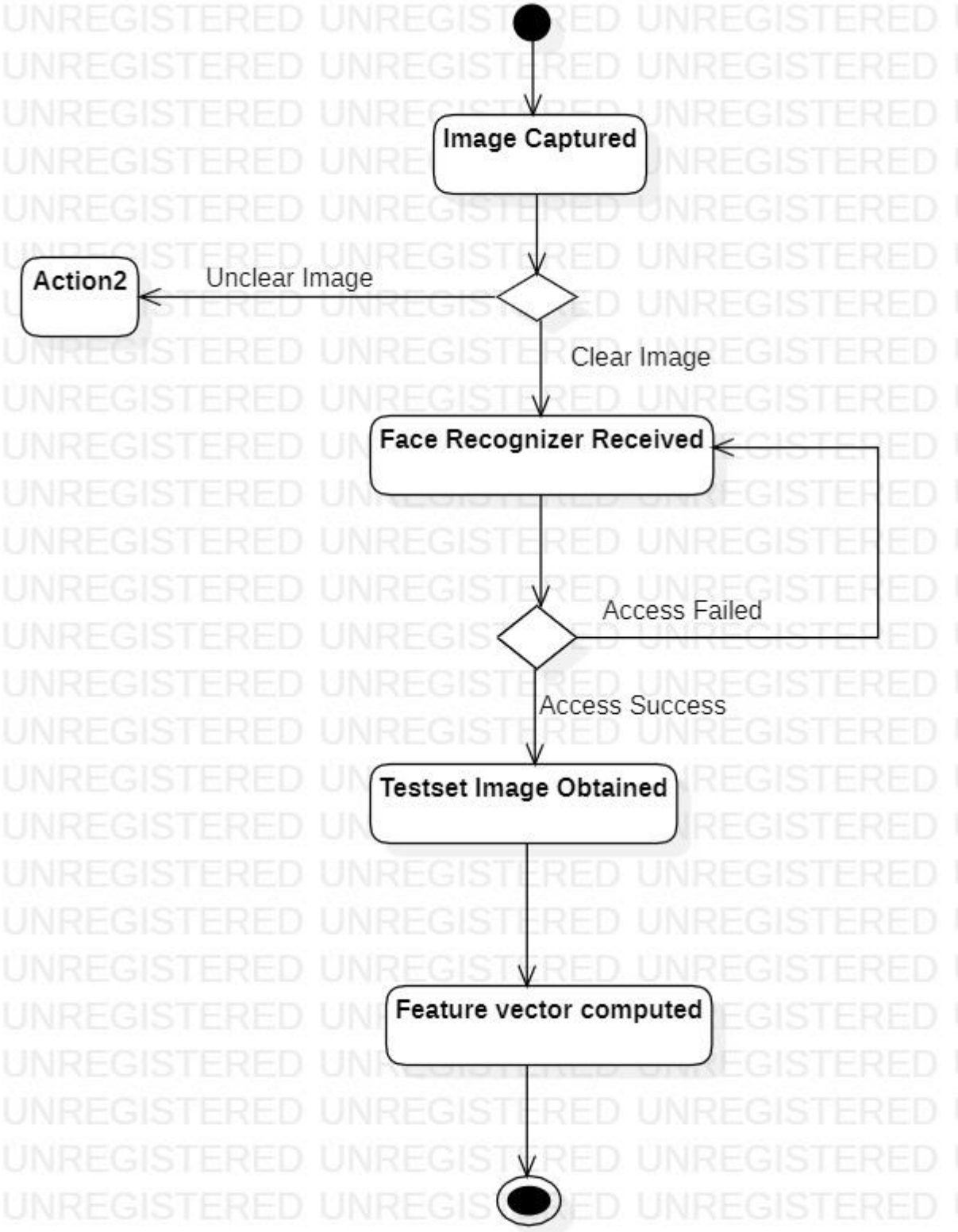
12.ACTIVITY DIAGRAM:

DETECT MOTION:



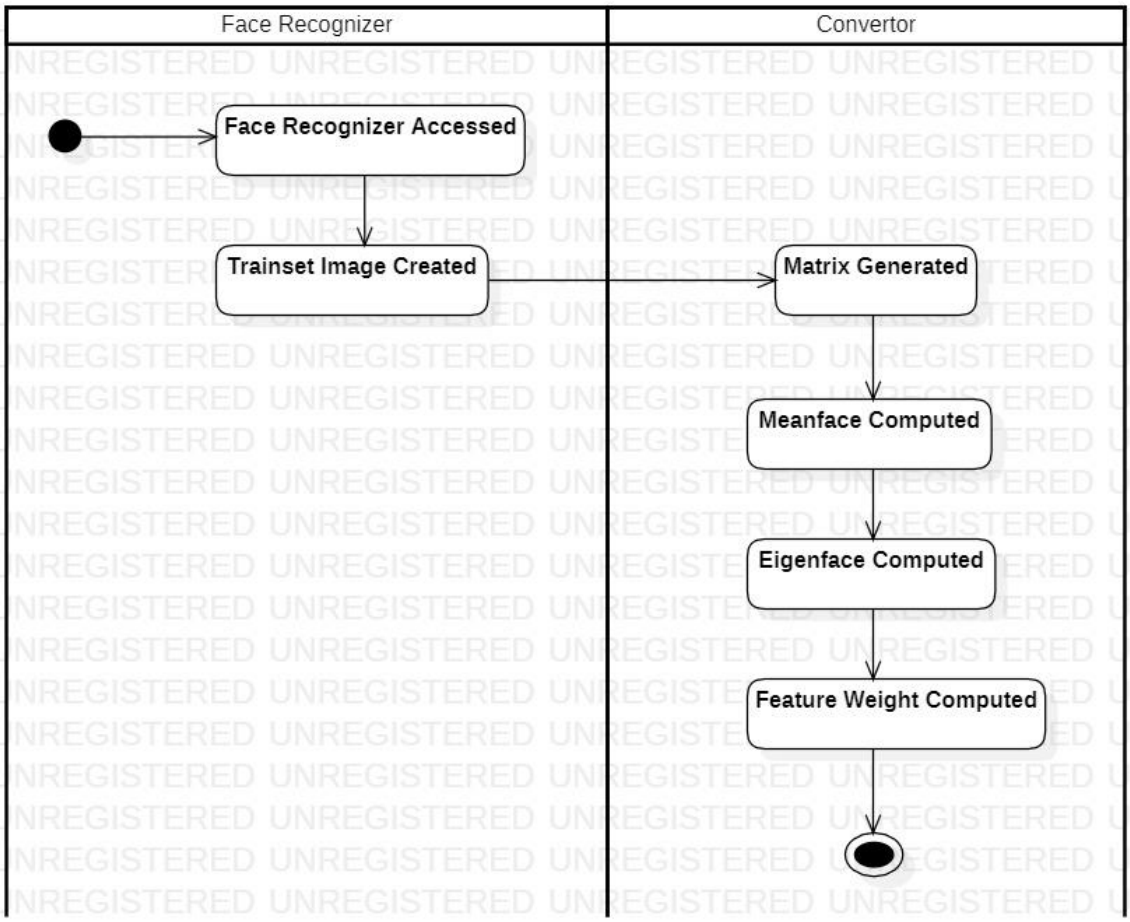
- This activity model uses a swim lane diagram and the two objects used here are **Homeowner** and **Pizo sensor.**
- From the initial point, the Homeowner activates **pizosensor.**
- Once it is activated the **input signal** will be **generated.**
- Then we have a activity of **validating signal** where there will be **choices** of **valid** and **invalid.**
- If the decision is **valid** then the **setup camera** activity is carried on and then it **exits** the control flow.
- If the decision is **invalid** then the **input signal** is generated again.

INTIALIZING FACERECOGNIZER:



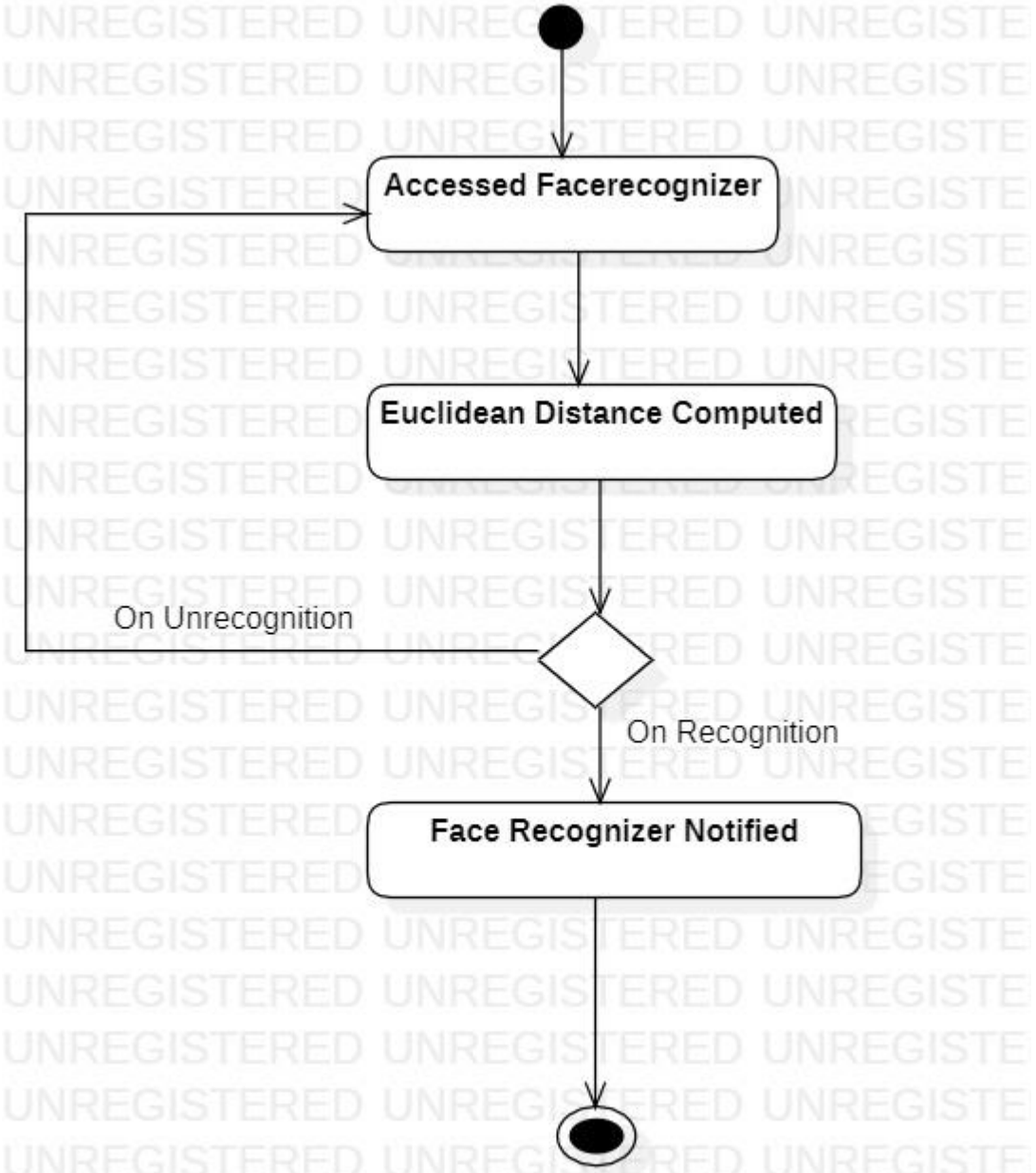
- We start with the **Image Captured** activity where we have 2 **choices** of **clear image** or **unclear image.**
- If the decision is **clear image** the **FaceRecognizer** is Accessed.
- Here we have **choices** of having **Access failed** or **Access success.**
- If the decision is **Access failed** the **FaceRecognizer** is **accessed** again.
- If the decision is **Access success** the **Testsetimage** is **obtained**.
- Once the **testset image** is **obtained** the **feature vector** is **computed** and then **exits** the control flow.

Compute Eigenface:



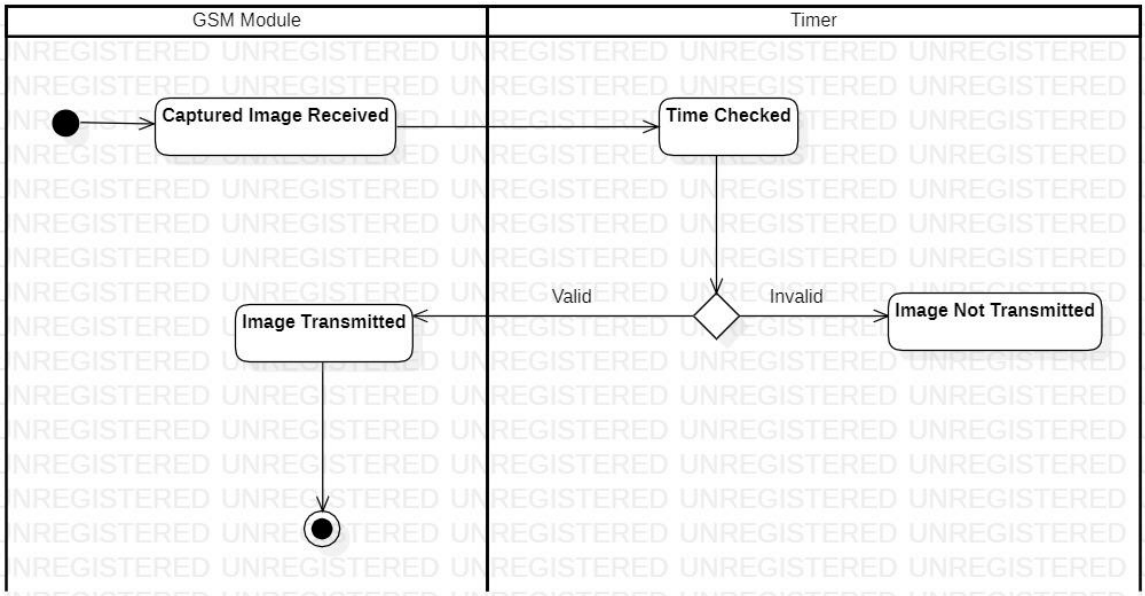
- This activity model uses a swim lane diagram and the two objects used here are **Face Recognizer** and **Convertor.**
- From the initial point the **FaceRecognizer** is **Accessed.** Then the **Trainsetimage** is **created.**
- Once the **trainset image** is **created** the **matrix** is **generated** by the convertor.
- Then the **meanface** is **computed** by the Converter. Using the generated matrix the meanface is computed. Using the meanface the Eigenface is computed. Finally the **feature weight** is **computed.**

FACERECOGNITION:



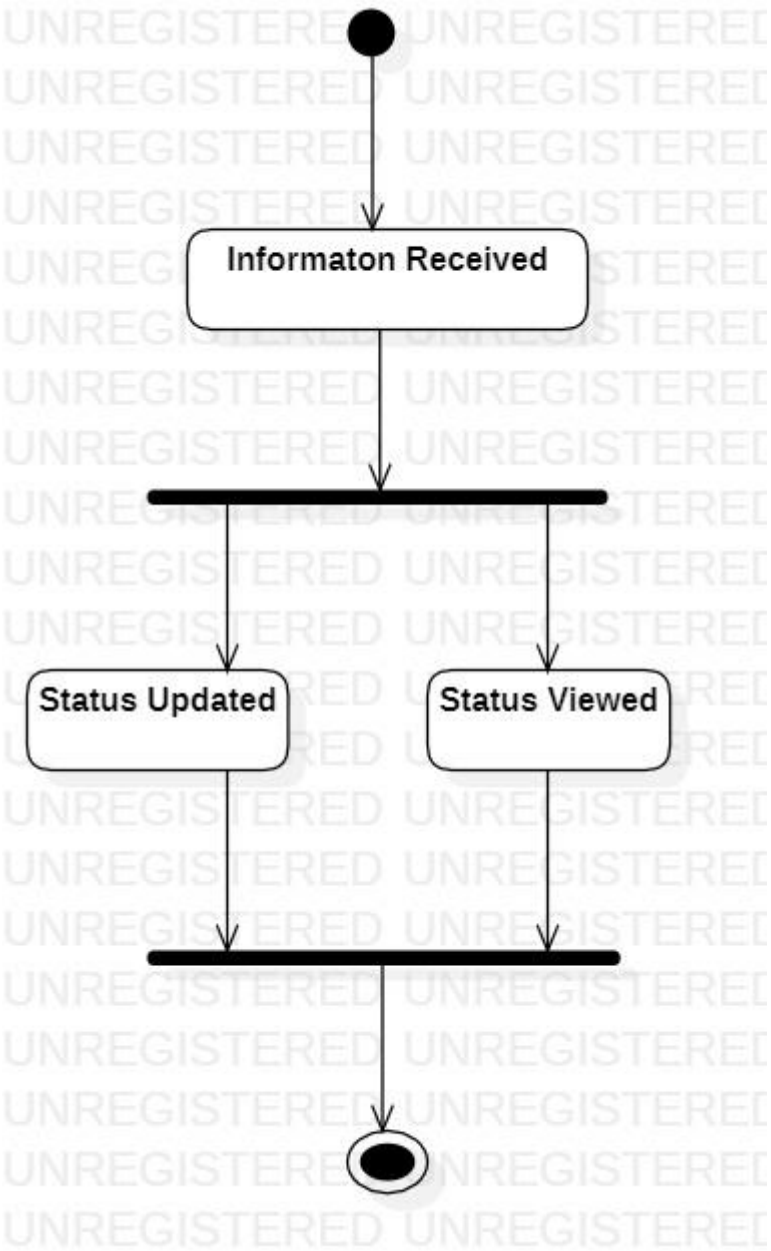
- From the initial point the **Accessed FaceRecognizer** should compute **Euclidean distance.**
- We have a **choice** of having **On recognition** or **On unrecognition**.
- If the decision is made **On Recognition** then the **FaceRecognizer** is **notified** and **exits** the control flow.
- If the decision is made **On Unrecognition** then this activity starts from the **beginning.**

TRANSMIT IMAGE:



- ✓ This activity model uses a swim lane diagram and the two objects used here are **GSMModule** and **Timer.**
- ✓ The **Captured Image** will be **received** to the **GSM module** and once the images are received the **GSMmodule** uses the timer to have a **Time checked** activity.
- ✓ Here we have 2 **choices** of having time **valid** or **invalid.**
- ✓ If the decision is **valid**, then the **GSMmodule** must **transmit image** to the **homeowner** and **exits** the control flow.
- ✓ If the decision is **invalid** then the **image** is **not transmitted**.

RESCUE:



- From the initial point the **information** is **received**.
- We have a **Fork** operation where the **status updated** and **status viewed**.
- Fork Operation:
    - Status updated
    - Status Viewed.

Then these 2 activities will be **joined** together and this.

CODE GENERATION:

CAMERA:

```
package IoT BASED ANTI-THEFT DETECTION AND ALARM SYSTEM;

import java.util.*;

/**
 *
 */
public class Camera {

    /**
     * Default constructor
     */
    public Camera() {
    }

    /**
     *
     */
    public int frame_height;

    /**
     *
     */
    public int frame_width;

    /**
     *
     */
    public int picture_quality;

    /**
     *
     */
    public string timestamp;

    /**
     *
     */
    public string filename;

     /**
     *
     */
    public void capture_faceimage() {
        // TODO implement here
    }

    /**
     *
     */
    public void create_imagefile() {
        // TODO implement here
    }

}
```

CHECKER:

```
import java.util.*;

/**
 *
 */
public class Checker extends Convertor {

    /**
     * Default constructor
     */
    public Checker() {
    }

    /**
     *
     */
    public double distance;


    /**
     * @return
     */
    public double compute_euclideandistance() {
        // TODO implement here
        return 0.0d;
    }

    /**
     *
     */
    public void find_closet image() {
        // TODO implement here
    }

}
```

CONTROLLER:

```
import java.util.*;

/**
 *
 */
public class Controller {

    /**
     * Default constructor
     */
    public Controller() {
    }

    /**
     *
     */
```

```
        public int position_x;

    /**
     *
     */
    public int position_y;

    /**
     *
     */
    public int brightness;



    /**
     *
     */
    public void move_up() {
        // TODO implement here
    }

    /**
     *
     */
    public void move_down() {
        // TODO implement here
    }

    /**
     *
     */
    public void move_left() {
        // TODO implement here
    }

    /**
     *
     */
    public void move_right() {
        // TODO implement here
    }

    /**
     *
     */
    public void increase_brightness() {
        // TODO implement here
    }

    /**
     *
     */
    public void decrease_brightness() {
        // TODO implement here
    }

}
```

CONVERTOR:

```
import java.util.*;

/**
 *
 */
public class Convertor {

    /**
     * Default constructor
     */
    public Convertor() {
    }

    /**
     *
     */
    public double Meanface;

    /**
     *
     */
    public double covariancematrix;

    /**
     *
     */
    public double eigenvector;

    /**
     *
     */
    public double featureweight;


    /**
     * @return
     */
    public double convertimagetomatrix() {
        // TODO implement here
        return 0.0d;
    }

    /**
     * @return
     */
    public double compute_meanface() {
        // TODO implement here
        return 0.0d;
    }

    /**
     * @return
     */
    public double compute_covariancematrix() {
        // TODO implement here
        return 0.0d;
    }
```

```
    /**
     * @return
     */
    public double compute_eigenface() {
        // TODO implement here
        return 0.0d;
    }

    /**
     * @return
     */
    public double compute_featureweight() {
        // TODO implement here
        return 0.0d;
    }

}
```

DISTANCE ANALYSER:

```
import java.util.*;

/**
 *
 */
public class Distanceanalyser {

    /**
     * Default constructor
     */
    public Distanceanalyser() {
    }

    /**
     *
     */
    public int distance;

    /**
     *
     */
    public void compute_mindistance() {
        // TODO implement here
    }

}
```

FACE RECOGNIZER:

```
import java.util.*;

/**
 *
 */
public class Face Recognizer {

    /**
     * Default constructor
     */
    public Face Recognizer() {
    }
```

```
    /**
     *
     */
    public void compare_faces() {
        // TODO implement here
    }

}
```

FLOOR MAT:

```
import java.util.*;

/**
 *
 */
public class Floor mat {

    /**
     * Default constructor
     */
    public Floor mat() {
    }

    /**
     *
     */
    private int length;

    /**
     *
     */
    private int breadth;

    /**
     *
     */
    private int thickness;



    /**
     * @return
     */
    private boolean isplaced() {
        // TODO implement here
        return false;
    }

}
```

GSM MODULE:

```
import java.util.*;

/**
 *
 */
public class Gsm module {

    /**
     * Default constructor
     */
    public Gsm module() {
    }

    /**
     *
     */
    public string phoneno;




    /**
     *
     */
    public void send_message() {
        // TODO implement here
    }

    /**
     *
     */
    public void receive_messages() {
        // TODO implement here
    }

}
```

HOMEOWNER:

```
import java.util.*;

/**
 *
 */
public class Homeowner extends Result {

    /**
     * Default constructor
     */
    public Homeowner() {
    }

    /**
     *
     */
    private string Name;
```

```
    /**
     *
     */
    private int Age;

    /**
     *
     */
    private string Phoneno;

    /**
     *
     */
    private string Gender;

    /**
     *
     */
    private string Address;
```

NEIGHBORHOOD:

```
import java.util.*;

/**
 *
 */
public class Neighborhood extends Rescue{abstract} {

    /**
     * Default constructor
     */
    public Neighborhood() {
    }

    /**
     *
     */
    public int doorno;

    /**
     *
     */
    public void Attribute1;



    /**
     *
     */
    public void getdetails() {
        // TODO implement here
    }

    /**
     *
     */
    public void updatestatus() {
```

```
        // TODO implement here
    }

}
```

PIZO-SENSOR:

```
import java.util.*;

/**
 *
 */
public class Pizo-sensor {

    /**
     * Default constructor
     */
    public Pizo-sensor() {
    }

    /**
     *
     */
    private void generate_inputsignal() {
        // TODO implement here
    }

    /**
     *
     */
    public void transmit_inputsignal() {
        // TODO implement here
    }

}
```

POLICESTATION:

```
import java.util.*;

/**
 *
 */
public class Policestation extends Rescue{abstract} {

    /**
     * Default constructor
     */
    public Policestation() {
    }

    /**
     *
     */
    public string Address;

    /**
     *
     */
    public long pincode;
```

```
    /**
     *
     */
    public void getdetails() {
        // TODO implement here
    }

    /**
     *
     */
    public void updatestatus() {
        // TODO implement here
    }

}
```

RASPBERRY – PI – CONTOLLER:

```
import java.util.*;

/**
 *
 */
public class Raspberry-pi-controller {

    /**
     * Default constructor
     */
    public Raspberry-pi-controller() {
    }

    /**
     *
     */
    private int Threshold;

    /**
     *
     */
    private int sensorvalue;

    /**
     *
     */
    private int ledoutput;









    /**
     * @return
     */
    private int Threshold() {
        // TODO implement here
        return 0;
    }

    /**
     * @return
```

```
     */
    private int sensorvalue() {
        // TODO implement here
        return 0;
    }

    /**
     * @return
     */
    private int ledoutput() {
        // TODO implement here
        return 0;
    }

    /**
     *
     */
    private void setup() {
        // TODO implement here
    }

    /**
     *
     */
    private void loop() {
        // TODO implement here
    }

}
```

RESCUE{ABSTRACT}:

```
import java.util.*;

/**
 *
 */
public class Rescue{abstract} {

    /**
     * Default constructor
     */
    public Rescue{abstract}() {
    }

    /**
     *
     */
    public string Name;

    /**
     *
     */
    public string Contactno;


    /**
     *
     */
    public void getdetails() {
        // TODO implement here
```

```
    }

}
```

RESULT:

```
import java.util.*;

/**
 *
 */
public class Result {

    /**
     * Default constructor
     */
    public Result() {
    }

    /**
     *
     */
    public string Status;




    /**
     *
     */
    public void create_status() {
        // TODO implement here
    }
```

ROUTER:

```
import java.util.*;

/**
 *
 */
public class Router {

    /**
     * Default constructor
     */
    public Router() {
    }

    /**
     *
     */
    private string Networktype;

    /**
     *
     */
    private int Throughput;
```

```
    /**
     *
     */
    private float Band;

    /**
     *
     */
    private double Range;

    /**
     *
     */
    private void Configure_router() {
        // TODO implement here
    }

}
```

TESTSET IMAGE:

```
import java.util.*;

/**
 *
 */
public class Test-set Image {

    /**
     * Default constructor
     */
    public Test-set Image() {
    }

    /**
     *
     */
    public double feature_vector;



    /**
     *
     */
    public void Read_testimage() {
        // TODO implement here
    }

    /**
     * @return
     */
    public double compare_featurevector() {
        // TODO implement here
        return 0.0d;
    }

}
```

TIMER:

```
import java.util.*;

/**
 *
 */
public class Timer {

    /**
     * Default constructor
     */
    public Timer() {
    }

    /**
     *
     */
    public int Time;



    /**
     *
     */
    public void check_time() {
        // TODO implement here
    }

}
```

TRAINSET IMAGE:

```
import java.util.*;

/**
 *
 */
public class Train-set Image {

    /**
     * Default constructor
     */
    public Train-set Image() {
    }

    /**
     *
     */
    public string train_setimage;



    /**
     *
     */
    public void save_image() {
        // TODO implement here
```

```
    }

    /**
     *
     */
    public void save_name() {
        // TODO implement here
    }

}
```