INFANT MONITORING SYSTEM WITH RFID AND IMAGE PROCESSING

# **ABSTRACT:**

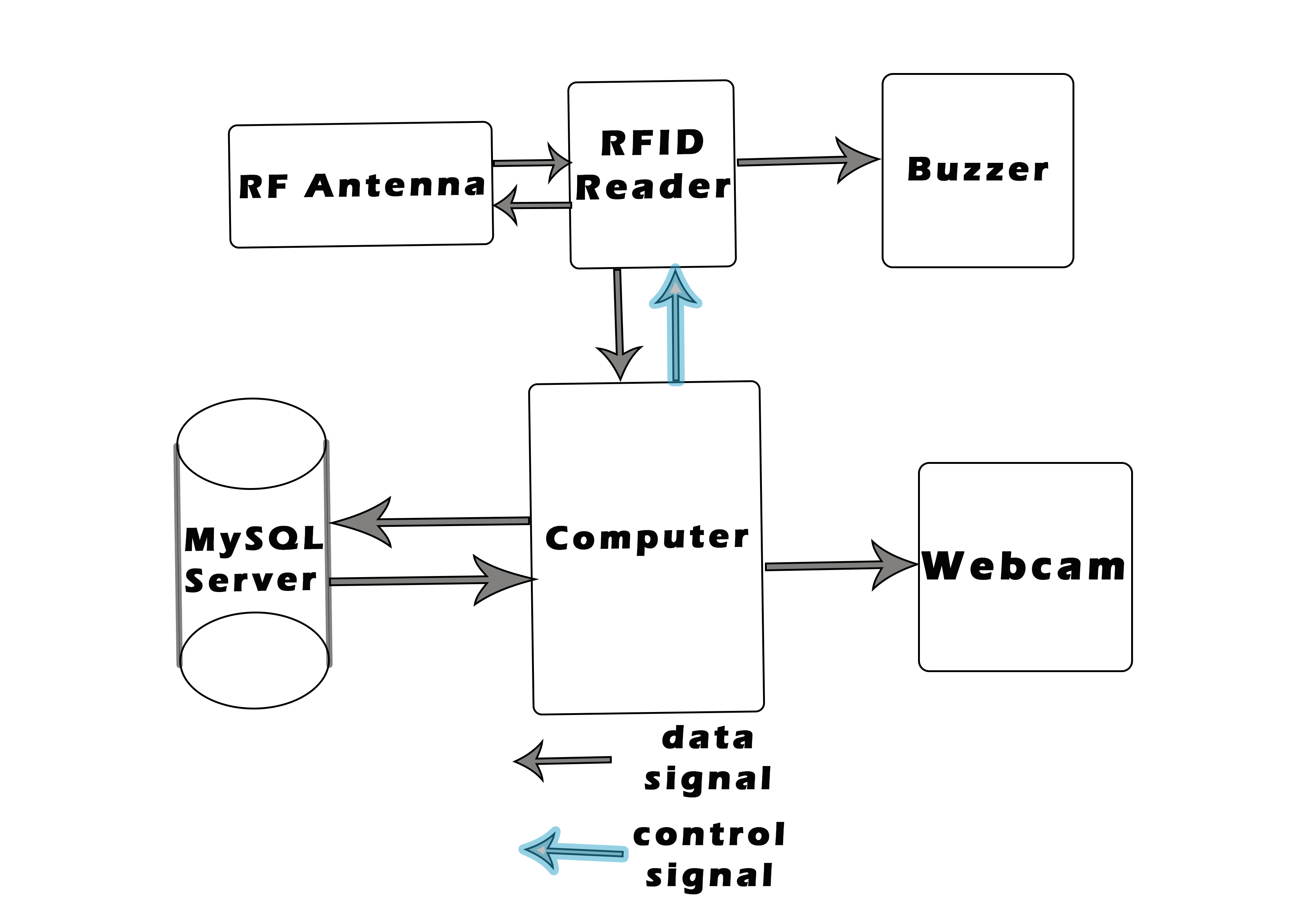
Babies are one of the most vulnerable patients in a hospital. Ensuring infant security is critical to not only the reputation of the hospital, but also to the peace of mind of everyone there, from nursing staff to new moms. This paper proposes a capable monitoring system for infants, involving the usage of **RFID Tags** and **Image Processing**. A prototype is developed which gives a reliable and efficient baby monitoring system that can play a vital role in providing better infant security. The RFID Tag is attached to the ankle of the new born baby to track its movements inside the hospital to ensure that baby is always attended by the right person and with the right mother. Also, if the tags have been misused, the facial recognition software helps us to find the right person for the detected RFID tag and if they are mismatched, the alarm will be triggered. This two-step verification helps in providing better security for the infants.

# **INTRODUCTION:**

The need for better security is promoting the use of patient monitoring systems in hospitals. Their contribution to better process management, superior flexibility and increased efficiency within hospitals is further underlining the appeal of these automated systems for patient monitoring systems. As awareness about the benefits of this security grows and the fact that creation of customized solutions for individual hospital requirements is possible will also boost implementation levels in the coming years.

The infant monitoring system consists of two steps of verification – the **RFID tag verification** followed by a **facial recognition** verification to make sure that the tags are always handled by the authorized individuals at all times. RFID is an acronym for “**radio-frequency identification**” and refers to the technology whereby digital data encoded in RFID tags or smart labels are captured by a reader via radio waves. RFID is similar to barcoding in that data from a tag or label are captured by a device that stores the data in a database. Every individual is given a RFID tag which contains a unique set of hexadecimal values. The infant tags are paired with the parent and attender tags. When there is a mismatch between the tags that are read, the buzzer is set. There have been many incidents of people misusing the tags wherein many individuals use the same tag to enter the secured area. In an effort to limit this, a facial recognition system is implemented which checks if the RFID tags are being held by the appropriate individuals. The facial recognition system checks the individuals against a stored set of images in the database and sets off an alarm in the event of any mismatch.

# **SYSTEM ARCHITECTURE:**



**BLOCK DIAGRAM**

# **RFID TAGS:**

In this system, the digital data encoded in RFID tags or smart labels are captured by a reader via radio waves. A radio-frequency identification system uses tags, or labels attached to the objects to be identified.

RFID tags can be either **passive, active** or **battery-assisted passive**. An active tag has an on-board battery and periodically transmits its ID signal. A battery-assisted passive (BAP) has a small battery on board and is activated when in the presence of an RFID reader.

Tags may either be **read-only**, having a factory-assigned serial number that is used as a key into a database, or may be **read/write**, where object-specific data can be written into the tag by the system user. Field programmable tags may be write-once, read-multiple; "blank" tags may be written with an electronic product code by the user.

RFID tags contain at least three parts: an **integrated circuit** that stores and processes information and that modulates and demodulates radio-frequency (RF) signals; a **means of collecting DC power** from the incident reader signal; and an **antenna** for receiving and transmitting the signal. The tag information is stored in a non-volatile memory. The RFID tag includes either fixed or programmable logic for processing the transmission and sensor data, respectively.

The RFID tags are given to the infants, parents and the attendants. Each tag has its own unique tag serial number (TagID) which makes it secure. To provide an additional layer of security, each TagID is encrypted before storing in the database to prevent attackers.

# **RFID READER:**

A radio frequency identification reader (RFID reader) is a device used to gather information from an RFID tag, which is used to track individual objects. Radio waves are used to transfer data from the tag to a reader. An RFID reader transmits an encoded radio signal to interrogate the tag. The RFID tag receives the message and then responds with its identification and other information such as the TagID. Since tags have individual serial numbers, the RFID system design can discriminate among several tags that might be within the range of the RFID reader and read them simultaneously. The RFID tag must be within the range of an RFID reader, which ranges from 3 to 15 feet, in order to be read.

The reader that we have used is the FX7500 Fixed RFID Reader by Zebra Technologies. The FX7500 comes in a sleek, attractive form factor with a low profile and compact footprint and supports up to 1200+ reads/sec in FM0 mode. It has higher sensitivity, improved interference rejection and echo cancellation. The reader is paired with a more flexible Linux-based network architecture with 512 MB Flash and 256 MB DRAM that integrates the tools and open-standards interfaces needed for fast, easy deployment with RFID and back-end applications. Integrated Power over Ethernet (PoE) lets you place the FX7500 where it is needed without installing extra outlets – ideal for large open areas. It provides EPC Global LLRP and RM interface support and also ships with comprehensive API support – .NET, C and Java which simplifies application development. The specs chart of the reader is also provided.

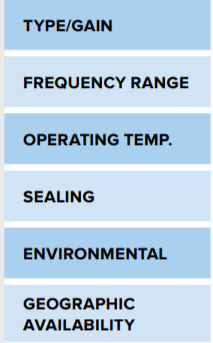
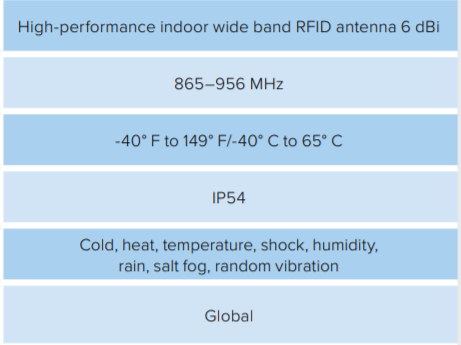


# **RF ANTENNA:**

Reader antennas convert electrical current into electromagnetic waves that are then radiated into space where they can be received by a tag antenna and converted back to electrical current. The RFID antenna propagates the wave in both vertical and horizontal dimensions. The field coverage of the wave and also its signal strength is partially controlled by the number of degrees that the wave expands as it leaves the antenna. While the higher number of degrees means a bigger wave coverage pattern it also means lower strength of the signal. Passive RFID tags utilize an induced antenna coil voltage for operation. This induced AC voltage is rectified to provide a voltage source for the device. As the DC voltage reaches a certain level, the device starts operating. By providing an energizing RF signal, a reader can communicate with a remotely located device that has no external power source such as a battery.

The antenna used in this implementation is Zebra Technologies’ AN480. The wide-frequency range enables the AN480 to be utilized in worldwide deployments, providing cost-efficiencies and a simplified RFID infrastructure. This antenna offers a low axial ratio to deliver a more uniform gain for superior performance. The AN480 single port antenna offers the flexibility and performance required to meet the needs of enterprises around the world. The AN480 can be installed throughout the enterprise in manufacturing and warehouse floor environments, or in the ceilings of a hospital for asset tracking applications.



# **MySQL Server:**

MySQL is an open-source relational database management system (RDBMS). MySQL uses a standard form of the well-known SQL data language. It works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc. It works very quickly and works well even with large data sets. MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

# **SOFTWARE REQUIREMENTS:**

RFID REQUIREMENTS:

* Install Java Version 8 from

<https://java.com/en/download/>

* Install Java SE Development kit 8u211 from

<https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

* Install Java Runtime Environment 8u211 from

<https://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>

* Download Apache NetBeans 9.0 (Binaries) from

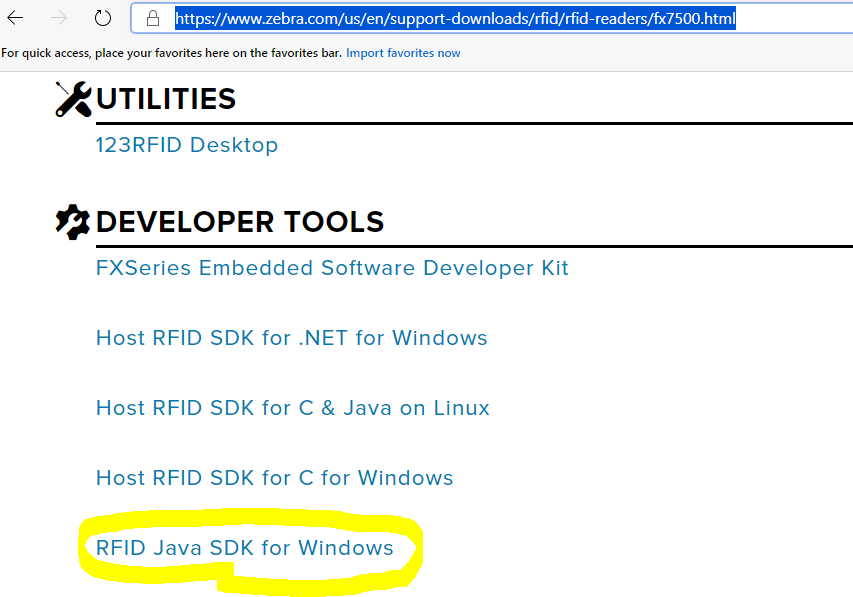
<https://netbeans.apache.org/download/nb90/nb90.html> and unzip it. Inside bin folder install netbeans64 application

* Install MySQL (full version) from

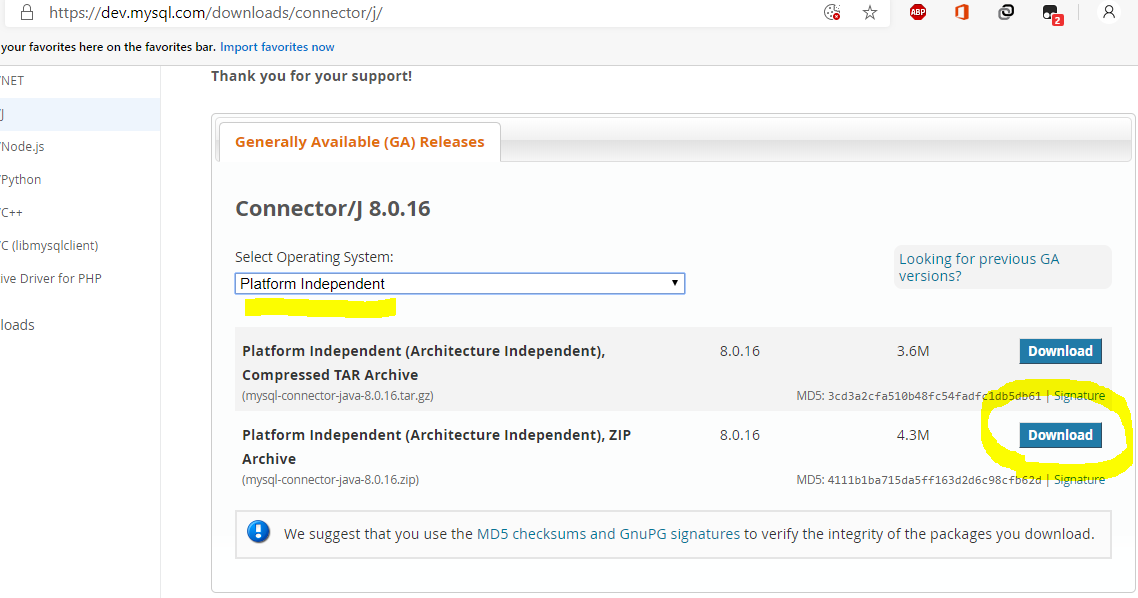
<https://www.mysql.com/downloads/>

* Download RFID java SDK from

<https://www.zebra.com/us/en/support-downloads/rfid/rfid-readers/fx7500.html>



* Download the MySQL to Netbeans Connector library from

[https://dev.mysql.com/downloads/connector/j/](https://dev.mysql.com/downloads/connector/j/)

* Some libraries for GUI inside Netbeans i.e. for calendar download the jar file from <http://www.java2s.com/Code/Jar/j/Downloadjcalendar14jar.htm>

IMAGE PROCESSING REQUIREMENTS:

* Python 3.6.8 (64bit AMD)
* Python libraries (pip installable):

1. numpy(1.16.1)
2. opencv-contrib-python
3. imutils
4. dlib (<https://files.pythonhosted.org/packages/0e/ce/f8a3cff33ac03a8219768f0694c5d703c8e037e6aba2e865f9bae22ed63c/dlib-19.8.1-cp36-cp36m-win_amd64.whl#sha256=794994fa2c54e7776659fddb148363a5556468a6d5d46be8dad311722d54bfcf> )
5. face\_recognition
6. cmake

To check if the software is installed correctly, enter the following command python --version in cmd. The version is returned if installation is done successfully. Rerun the setup and try the modify or repair options if the installation is unsuccessful.

To pip install the module, use the following command : python -m pip install module\_name==version. For dlib, use the link provided instead of the module name. Version number is not mandatory.

To check if the module is installed correctly, enter the following command python -c "import module\_name; print(module\_name.\_\_version\_\_)" which should return the module version upon successful installation.

# **FACIAL RECOGNITION WORKING:**

Face recognition is really a series of several related problems:

1. First, look at a picture and find all the faces in it
2. Second, focus on each face and be able to understand that even if a face is turned in a weird direction or in bad lighting, it is still the same person.
3. Third, be able to pick out unique features of the face that you can use to tell it apart from other people— like how big the eyes are, how long the face is, etc.
4. Finally, compare the unique features of that face to all the people you already know to determine the person’s name.

Computers are not capable of this kind of high-level yet, so we have to teach them how to do each step in this process separately. We need to build a **pipeline** where we solve each step of face recognition separately and pass the result of the current step to the next step. In other words, we will chain together several machine learning algorithms.

STEP 1: CREATING THE IMAGE DATASET

The dataset plays an important part in the success of any face recognition algorithm. The dataset is the input collection of people’s images that the software uses to work with. The images in the dataset needs to be as different as possible so as to increase the software’s accuracy. Images with different lightings and different facial angles may be preferred.

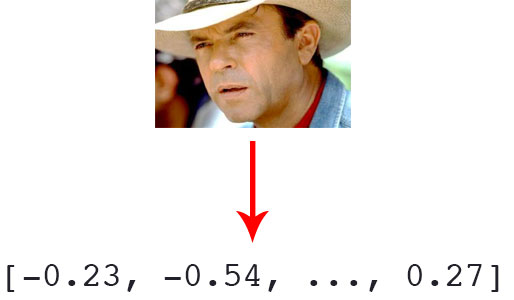
We use **haar-cascade** files for face detection. OpenCV already contains many pre-trained classifiers for face, eyes, smiles, etc. These XML files are stored in the Python\Python36\Lib\site-packages\cv2\data. First, go to the directory containing the python scripts in cmd and run it in the following format

python python\_script\_name.py --cascade location\_of\_xml\_file --output output\_folder\_where\_the\_dataset\_is\_stored

Our dataset algorithm takes 100 pictures of the individual in 10 seconds, and the individuals are instructed to vary the facial angles and expressions.

STEP 2: ENCODING THE IMAGES:

Before we can recognize faces in images and videos, we first need to quantify the faces in our training set. Keep in mind that we are not actually training a network here — **the network has already been trained to create 128-d embeddings** on a dataset of ~3 million images. We certainly could train a network from scratch or even fine-tune the weights of an existing model but that is more than likely overkill for many projects. Furthermore, we would need a lot of images to train the network from scratch. Instead, it’s easier to use the pre-trained network and then use it to construct 128-d embeddings for each of the 218 faces in our dataset.



Our algorithm encodes each images in the dataset using the HOG algorithm in binary format and dumps the data in a pickle file. The pickle file is then passed as input to the actual face recognition program which uses the pickled data to label the images. To run the python script, go to the directory containing it and enter the following command:

python python\_script\_name.py --dataset location of dataset --encodings location of pickle file

STEP 3: FACIAL RECOGNITION

Many facial recognition algorithms are available now and we are using a method called **Histogram of Oriented Gradients** (**HOG**) which is reliable and efficient to work in real time. To detect faces, first, we’ll look at every single pixel in our image one at a time. For every single pixel, we want to look at the pixels that directly surrounding it. If you repeat that process for every single pixel in the image, you end up with every pixel being replaced by an arrow. These arrows are called gradients and they show the flow from light to dark across the entire image. If we analyse the pixels directly, really dark images and really light images of the same person will have totally different pixel values. But by only considering the direction that brightness changes, both really dark images and really bright images will end up with the same exact representation. But saving the gradient for every single pixel gives us way too much detail. It would be better if we could just see the basic flow of lightness/darkness at a higher level so we could see the basic pattern of the image. To do this, we’ll break up the image into small squares of 16x16 pixels each. In each square, we’ll count up how many gradients point in each major direction (how many point up, point up-right, point right, etc…). Then we’ll replace that square in the image with the arrow directions that were the strongest. The end result is we turn the original image into a very simple representation that captures the basic structure of a face in a simple way. To find faces in this HOG image, all we have to do is find the part of our image that looks the most similar to a known HOG pattern that was extracted from a bunch of other training faces.

All we need to do is train a classifier that can take in the measurements from a new test image and tells which known person is the closest match and a simple SVM Classifier is used to perform the classification.

Adam Geitgey’s face\_recognition module is extremely easy to work with and performs most of these actions with ease. Every individual having a RFID tag will have their images collected, so that the model can be trained to identify them i.e., every RFID tag is paired with the images of the person carrying it. When the tag is read, the program immediately tests for verification of the holder’s face so that the tags are not misused by others. Upon mismatch, the buzzer is set. To run the python script, go to the directory containing the script and enter the following command:

python python\_script\_name.py --encodings location of pickle file --output location of output video