



project report on

Medical Box



Medox

(Final year project)

Submitted in partial fulfillment, of the requirement for the award of the
Degree of Bachelor in
COMPUTERS AND SYSTEMS ENGINEERING

Submitted by:

- Ahmed Ismail El-Sayed Ismail
- Amira Ibrahim Soliman
- Gelan Elhady Elsayed Mohamed
- Mahmoud Mohamed Mahmoud Abdel Maksoud
- Mostafa El-Sayed Gohary selim

Supervised by:

Dr. Hazem I. Shehata

July, 2017

Faculty of Engineering
Zagazig University

1. project overview.....	3
1.1. Abstract.....	3
1.2. Introduction.....	4
I. Problem definition.....	4
II. approach and tools/techniques.....	4
III. overview of system modules.....	5
1.3. Introduction to IOT.....	6
1.4. Box history.....	9
2. Box design.....	11
2.1. Introduction.....	11
2.2. Pill dispensing mechanism history.....	13
I. Tube and plate.....	14
II. Rotating plate.....	15
III. Suction mechanism.....	16
3. Block diagram.....	17
4. Mechanism overview.....	18
4.1. Tray (first motor).....	19
4.2. Nozzle apparatus (second motor).....	20
4.3. Motor Selection.....	21
5. Hardware Components.....	22
5.1. Stepper Motor.....	22
5.2. Motor driver.....	28
5.3. Vacuum pump.....	30
5.4. Pump control.....	31
5.5. Infrared IR (safety).....	31
5.6. Lock “servo motor”.....	32
5.7. Motors initialization.....	35
5.8. Power supply.....	36
5.9. Controller (Arduino Uno).....	37

6. Software Introduction.....	38
7. Connection.....	40
7.1. Flow chart.....	40
7.2. Rejected connection schemas.....	42
7.3. Firebase.....	43
7.4. Firebase Components description.....	45
I. Firebase Real-time Database.....	45
II. Firebase Authentication.....	47
III. Firebase Cloud Messaging.....	49
IV. Cloud Functions for Firebase.....	51
7.5. Google FIT.....	53
8. Box Client.....	55
8.1. Introduction.....	55
8.2. Raspberry-Pi.....	57
9. Mobile Client.....	58
9.1. Introduction.....	58
9.2. Android Application.....	59
9.3. Android app features include.....	60
9.4. Android Screens.....	61
10. Appendix.....	69
10.1. Solved problems.....	69
10.2. Known issues.....	72
10.3. Future Work.....	73
10.4. Design parts.....	74
10.5. Project Links.....	75
10.6. License.....	76
10.7. Third party Notice.....	78
10.8. References.....	80

1. Abstract

This report is a documentation of the final year graduation project in Electrical Engineering Zagazig University.

The purpose of this project is to design, build and control a Medical box which offer an easy and handy way to help caregivers take care of senior citizens and keep track of their medication and keep them organized,

also will give them a fast way to ask for help in emergency situations and a way for caregiver to track vital signs and location of senior citizen.

The project is divided into five main parts

- Hardware,
- Control,
- Connection,
- Box Client,
- and Mobile Clients.

1. Project overview

2. Introduction:

I. Problem definition

Presently, many people are dependent upon medications in their daily lives. An increasing number, especially the elderly, are required to take multiple different types of medications at different times throughout the day. This can become a complicated procedure for anyone, causing missed doses, incorrect doses, and potentially life threatening mistakes.

Senior citizens who live alone can't organize their medications and the number of pills that must take, and sometimes not even remember. Also caregivers don't have an easy way to take care of them.

II. approach and tools/techniques

medicine dispensing machines that can solve these problems currently exist but expensive and not efficient. Clearly, the average household cannot afford this luxury. Based on this situation, a user friendly, affordable, semi-compact device to dispense medicine at home will be developed.

Medical box is to offer an easy and handy way to help caregivers take care of senior citizens and keep track of their medications and keep them organized, also will give them a fast way to ask for help in emergency situations.

The project name "Medox" came from the project function "medical box", the project consists of two main parts, the pills dispenser box, and the mobile applications. The pills dispenser box contains an automated pills dispenser that will make the process of taking an individual's complex pill prescription as automated as possible. The pills dispenser will be powered by a standard 220 VAC, and the second part is the mobile application that represents the user interface.

1. Project overview

III. overview of system modules

The project is divided to five main parts

- Hardware [box]
- Control [microcontroller]
- Connection [Cloud]
- Box Client [Raspberry-Pi]
- Mobile Clients [Android app]

Hardware part: is the implementation of box body and whatever mechanics needed for handling the whole medicines dispensing process.

Control part: is the software functions which will control the whole hardware processes.

Connection: a way to connect caregiver, senior citizen and box with each other.

Box Client: is the main program which will receive the schedule from the user and carry it out.

Mobile Clients: is the Application which will be used to control the server remotely and will also receive all the notifications from the server.

3. Introduction to IOT:

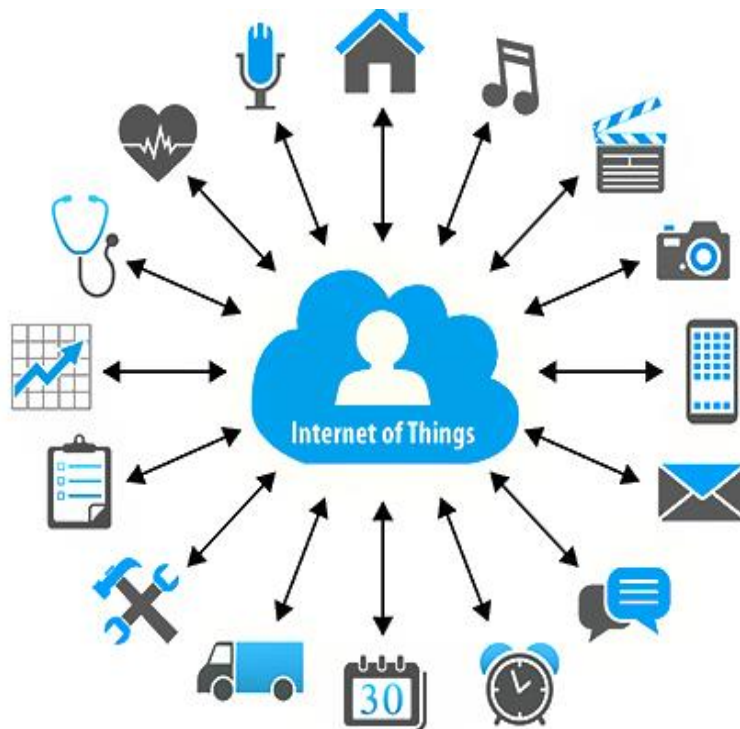


Fig (1) IOT

IOT stands for internet of things. Internet of Things is a world-wide network of interconnected physical objects uniquely addressable that contain embedded technology to communicate and sense or interact with their internal state or the external environment.

In 2013 the Global Standards Initiative on Internet of Things (IOT-GSI) defined the IOT as “a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies”. The IOT allows objects to be sensed or controlled remotely across existing network infrastructure, creating opportunities for more direct integration of the physical world into computer-based systems, and resulting in improved efficiency, accuracy and economic benefits in addition to reduced human intervention.

1. Project overview

IOT has many applications that makes life much easy and those are some of them are:

- Media
- Environmental monitoring
- Infrastructure management
- Manufacturing
- Agriculture
- Energy management
- Medical and healthcare
- Building and home automation
- Transportation

In our project we talk about “Medical and healthcare”.



Fig (2) IOT Medical Applications

Challenges in developing IOT solutions

1. Security:

- Securing networks
- Encrypted data lifecycle
- Firmware updates in the field
- Authentication persons/devices
- Log and track of collected data

2. Connectivity:

- Bandwidth Management
- SIM lifecycle Management

3. Communications:

- Many protocols:
- Old or customs protocols
- Open source protocols
- Different payload formats

4. Bi-directionality:

- Data collection
- Data routing
- Data stream guarantee
- Device Control

5. Complexity:

- Handling millions of connections
- Highly available Infrastructure and services
- Processing and storing high volume of Data
- Set business rules and alerts

4. Box history

Types of Medical Pill Organizer

1. Medical Pill Organizer (Plastic)

- the Seniors should distribute pill by pill in the box
- the Seniors don't know if the patient takes the pills or not
- hard to modify the Schedule
- hard to take extra pills
- limit by 7 days
- there is no alarm to the patient with the medicine time



Fig (3) Medical Pill Organizer

1. Project overview

2. Electronic Medication Organizer with Alarm Reminders

- the Seniors should distribute pill by pill in the box
- the Seniors don't know if the patient takes the pills or not
- hard to modify the Schedule
- hard to take extra pills



Fig (4) Electrical Medical Pill Organizer

2. Box design

1. Introduction

The encasement was designed to enclose the subsystems of the machine while minimizing the volume of the case as much as we could.

It has been made from black acrylic sheet with press fit design to make assembling easier and supported by metal screws made of steel to add sturdiness to the structure.

The base of the frame is a (30 cm * 30 cm) square, and the frame is 30 cm tall with a total volume 27000 square centimeters.

All The top panel of the frame will be hinged and held down with a lock which will allow access to the pills, also the dispensed pills locker has a lock to open only at scheduled times.

Pictures below are the design of the encasement and the layout of the subsystems within the Box.

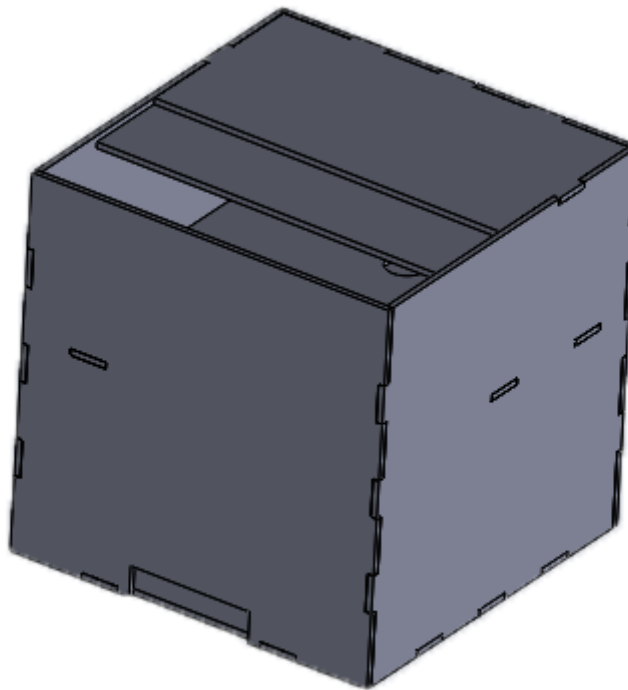


Fig (5) Assembled Box

2. Box design

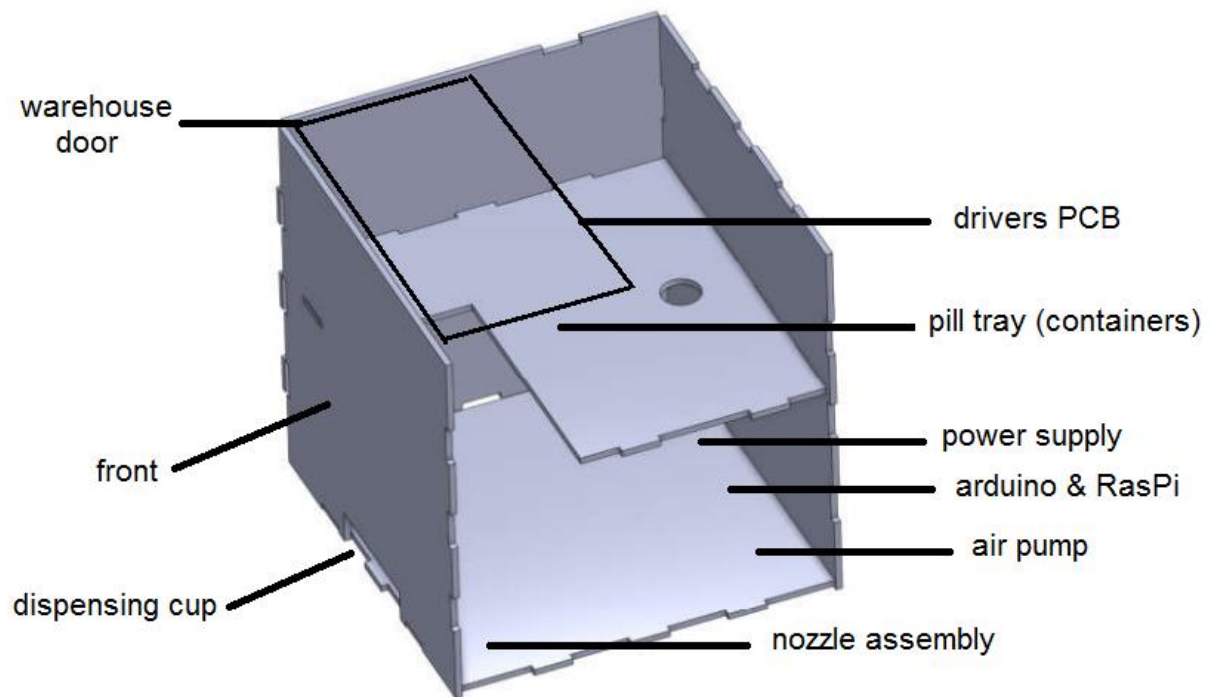


Fig (6) Box parts

Note: Pictures of each part are added to the appendixes.

2. Pill dispensing mechanism History

Previous designs

The main task of the pill dispenser mechanism is to move the selected pill from its container to the locker.

The major mechanism design constraints are

1. mechanism has to deal with different pill shapes and masses.
2. mechanism have to move one and only one pill by the time.
3. feeding the mechanism with pills must be easy.

We have designed three mechanisms before we apply the current one.

The next three sections describe the three different designs, there advantages and disadvantages, and why we did choose the current mechanism.

2. Box design

I. Tube and plate:

It consists of tube for each pill type opened from both sides and plate under it that can be moved forward by a motor to let the pill fall and return back. The design is illustrated in figure below.

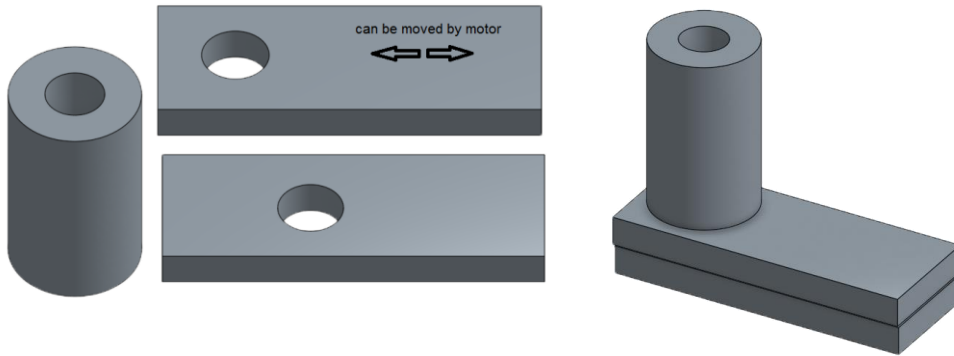


Fig (7) Tube and plate

Pros:

- Feeding the tubes with pills is easy for the user.
- Ease of implementation.

Cons:

- to make sure that only one pill would fall we have to customize each tube for only one pill shape.
- each pill tube requires separate motor (more motors)

2. Box design

II. Rotating plate:

The design consists of a fixed circular plate that has a small hole to let the pill pass through, and a rotating cylinder that is divided into small lockers to fit one pill.

The rotating cylinder pushes the pill to fall in the cup through the hole in the fixed plate. The design is illustrated in figure below.

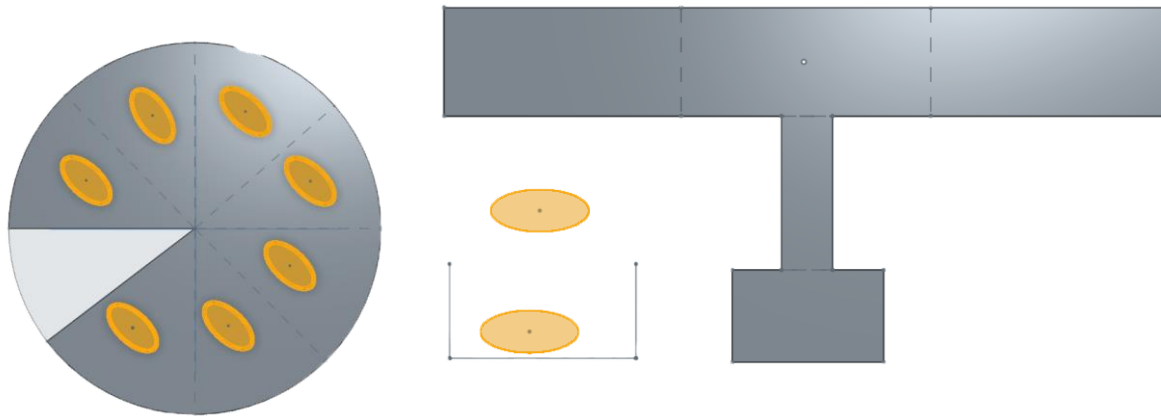


Fig (8) Rotating plate

Pros:

- Effective with all pill shapes and masses.
- Ease of implementation.

Cons:

- Require one motor for each pill type.
- It takes a lot of time from the user to separate each pill in its place (not user friendly).

2. Box design

III. Suction mechanism:

This mechanism consists of circular rotating disk (pill tray) which the pill repositories are attached to, depends on the power of suction to pick the required pill from the tray, then, moves to the locker to let the pill fall into. The design is illustrated in figure below.

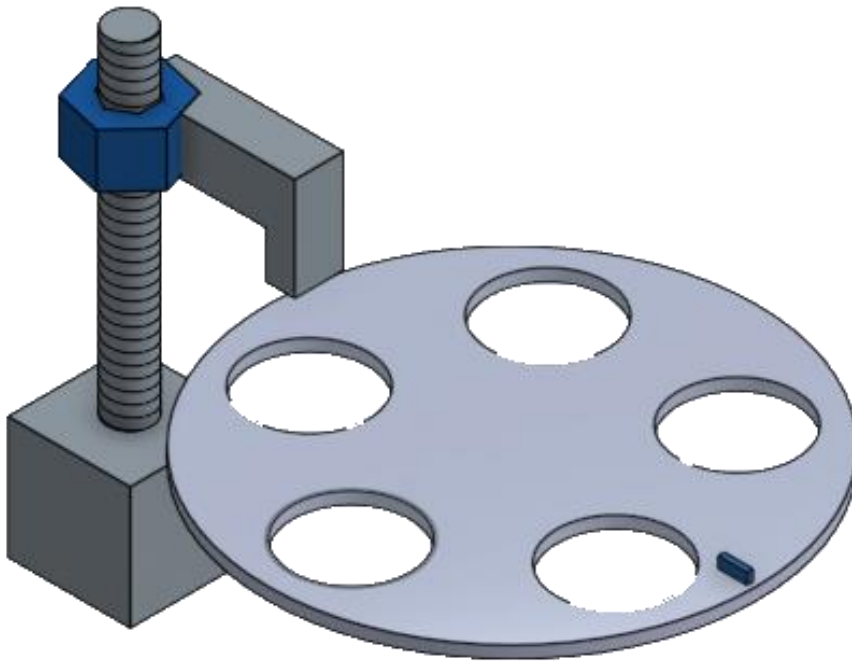


Fig (9) Suction mechanism

This mechanism was the most reliable, easy to use, and effective one so it was used in the project. The design is completely explained in details in the next parts of this chapter.

3. Block diagram

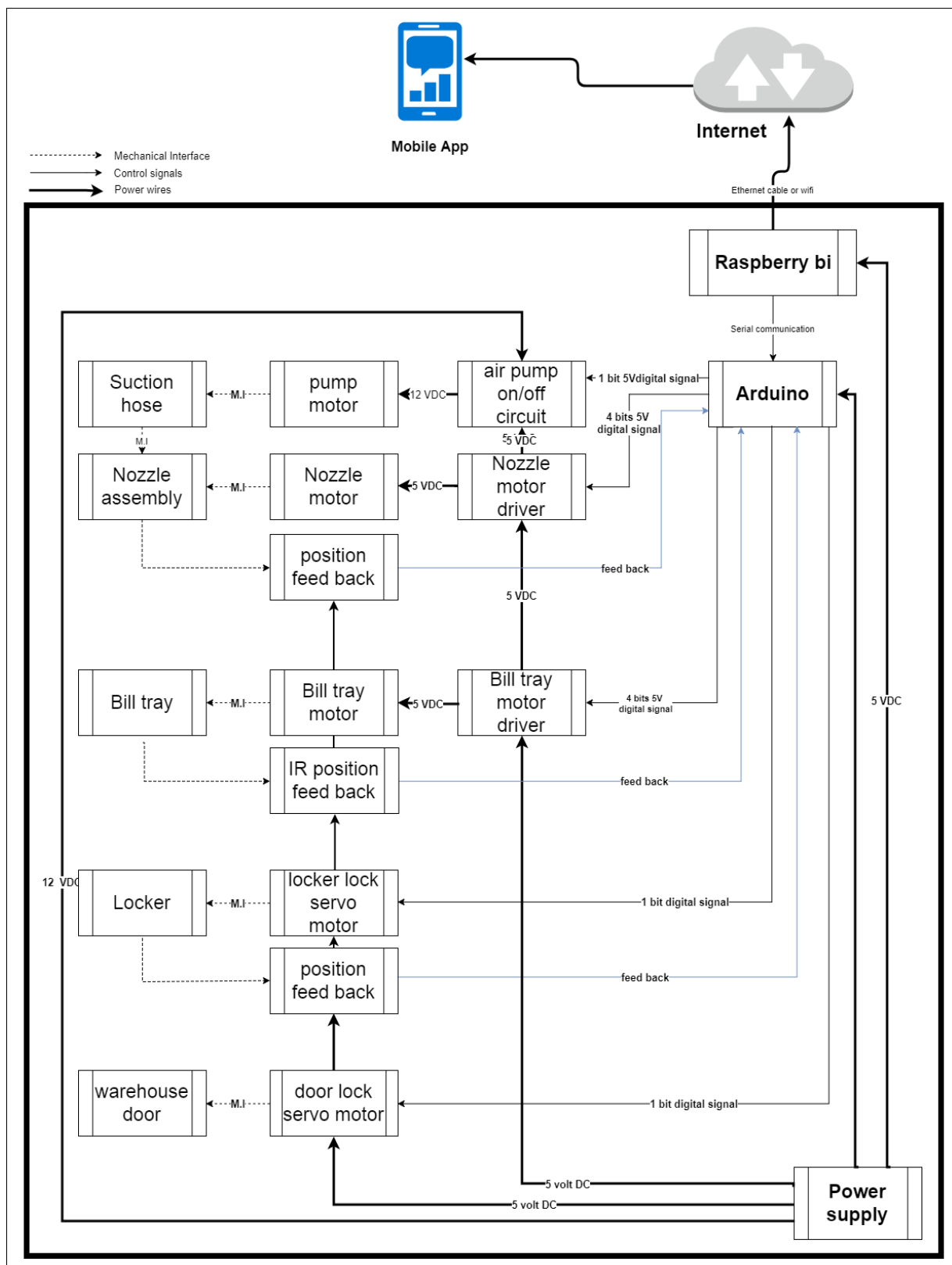


Fig (10) Block Diagram

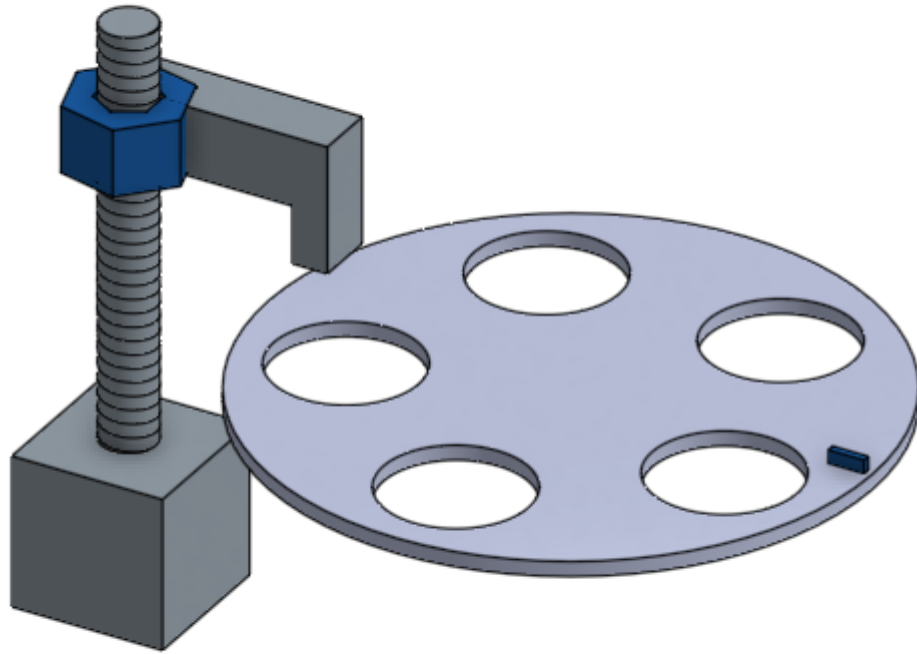


Fig (11) Tray and hose holder

4. Mechanism overview

1. Tray (first motor)

Pill tray is a plate with diameter of (20cm), also has a 4 pill bins and a hole through which pills will be dropped into warehouse. Each bin hole will be (4.8cm) diameter as will the hole. The angle between the centers of two consecutive bins is (72 degrees).

Since the vacuum nozzle will only translate vertically, the tray will be rotated by a stepper motor to position the pill bins beneath the vacuum such that a bin will be centered with the nozzle's axis of translation. The vacuum nozzle will lower into a pill bin, grab a pill, and then rise back up holding the pill. The tray will then rotate so that the hole will be located beneath the vacuum, and the vacuum will disengage, dropping the pill through the hole. The warehouse will be positioned in a way that it rests beneath the vacuum nozzle. Then when the pill dropped through the hole it falls into warehouse.

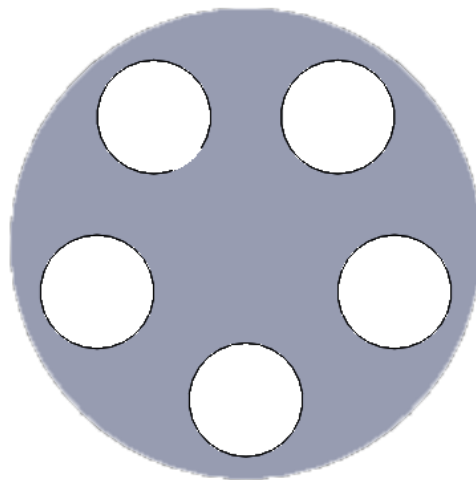


Fig (12) Tray

4. Mechanism overview

2. Nozzle apparatus (second motor)

the major function is to translate the rotation motion to translational motion also to hold the Hose, that will go down to pick the pill and go back to drop it in the locker.

each revolution = 0.7 cm

It is a stepper motor connected with a nozzle by a nail this nozzle moves up and down in a vertical motion to pulls up the pills from the tray.

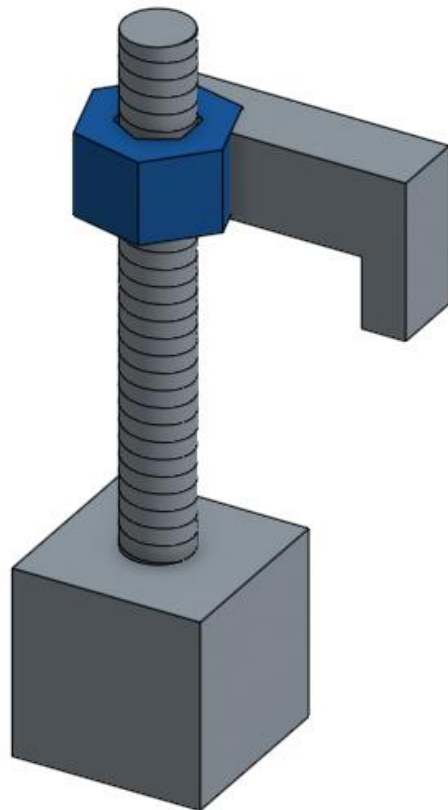


Fig (13) Up/Down motor

4. Mechanism overview

3. Motor Selection:

In the current design plan, two motors are necessary: one to rotate the pill tray and one to drive the pinion which will drive the translation of the nozzle apparatus.

The main issue involved finding motors within the designated price range that will provide sufficient torque and precision for the required applications. For this reason, it was decided from the beginning that stepper motors would be incorporated. Following this decision, calculations were carried out to determine what torque output would be required from each motor.

When the motors were initially selected, errors in calculating the torque requirements for both motors led to the selection of motors providing significantly higher output torque than was called for by design criteria. An immediate consideration was returning the motors for a refund and finding smaller motors; however, the cost of returning the motors would offset any positive gain from incorporating smaller, cheaper motors.

5. Hardware components

1. Stepper Motor

What is a stepper motor?

A stepper motor is a brushless, synchronous electric motor that converts digital pulses into mechanical shaft rotations.

The motor's position can then be commanded to move and hold at one of these steps without any feedback sensor (an open-loop controller), as long as the motor is carefully sized to the application in respect to torque and speed.

As the digital pulses from the controller increase in frequency, the stepping movement converts into a continuous rotation with the velocity of the rotation directly proportional to the frequency of the control pulses. Stepper motors are widely used because of their low cost, high reliability, and high torque at low speeds. Their rugged construction enables you to use stepper motors in a wide environmental range.

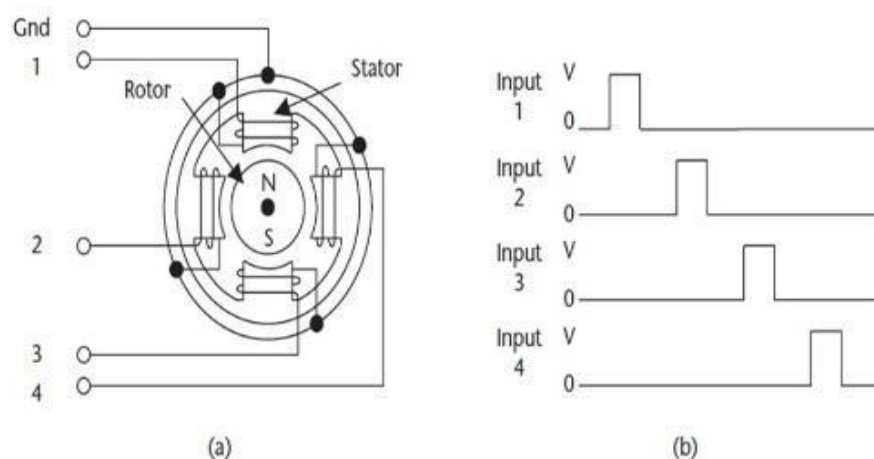


Figure 13.22 Four-phase stepper motor: (a) motor layout, and (b) driving waveforms.

Fig (14)

Following this decision, calculations were carried out to determine what torque output would be required from each motor.

Advantages of stepper motor:

High torque at startup and low speeds

The rotation angle of the motor is proportional to the input pulse.

Precise positioning and repeatability of movement since good stepper motors have an accuracy of 3 – 5% of a step and this error is non-cumulative from one step to the next.

Excellent response to starting/stopping/reversing.

5. Hardware components

The motors response to digital input pulses provides open-loop control, making the motor simpler and less costly to control.

It is possible to achieve very low-speed synchronous rotation with a load that is directly coupled to the shaft.

A wide range of rotational speeds can be realized as the speed is proportional to the frequency of the input pulses.

Limitations:

- **Low Efficiency** – Unlike DC motors, stepper motor current consumption is independent of load. They draw the most current when they are doing no work at all. Because of this, they tend to run hot.
- **Limited High Speed Torque** - In general, stepper motors have less torque at high speeds than at low speeds. Some steppers are optimized for better high-speed performance, but they need to be paired with an appropriate driver to achieve that performance.
- **No Feedback** – Unlike servo motors, most steppers do not have integral feedback for position. Although great precision can be achieved running ‘open loop’. Limit switches or ‘home’ detectors are typically required for safety and/or to establish a reference position.

5. Hardware components

Stepper motor types:

- 1. Unipolar stepper motors:** The unipolar stepper motor operates with one winding with a center tap per phase. Each section of the winding is switched on for each direction of the magnetic field. Each winding is made relatively simple with the commutation circuit; this is done since the arrangement has a magnetic pole which can be reversed without switching the direction of the current.

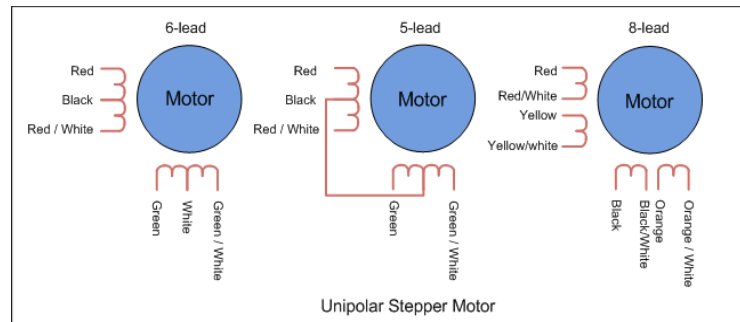


Fig (15)

- 2. Bipolar stepper motors:** With bipolar stepper motors there is only a single winding per phase. The driving circuit needs to be more complicated to reverse the magnetic pole, this is done to reverse the current in the winding.

This is done with “H-bridge” arrangement, however there are several driver chips that can be purchased to make this much simple task.

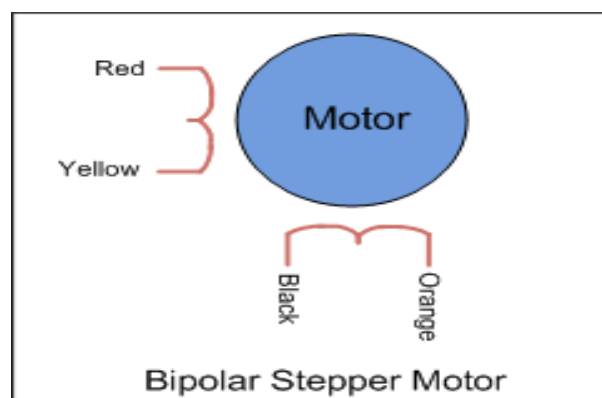


Fig (16)

5. Hardware components

Step Modes:

Stepper motor "step modes" include full step, half step, and micro step. The type of step is dependent on the stepper motor driver controlling the stepper motor. Many stepper motor controllers are multi-step capable (usually adjusted by switch setting).

- 1. Full Step:** Standard hybrid stepping motors have 200 full steps per revolution. If you divide the 200 steps into the 360 degrees of rotation you get 200 1.8 degree steps. Normally this is achieved by energizing both windings while alternately reversing the current, meaning one pulse from the driver is equal to one full step on the step motor.

Full mode				
sequence	Coil 1	Coil 2	Coil 3	Coil 4
Step 1	1	0	0	0
Step 2	0	1	0	0
Step 3	0	0	1	0
Step 4	0	0	0	1

5. Hardware components

2. **Half Step:** Half Step means that the stepping motor is rotating at 400 steps per revolution ($0.9 \text{ degree steps} \times 400 = 360 \text{ degrees}$). First one winding is energized and then two windings are alternately energized. This will cause the rotor of the stepping motor to move at half the distance (0.9 degrees). In half-step mode, a typical stepper motor provides about 30% less torque, but it provides a smoother motion than it would in full-step mode.

Half mode				
sequence	Coil 1	Coil 2	Coil 3	Coil 4
Step 1	1	0	0	0
Step 2	1	1	0	0
Step 3	0	1	0	0
Step 4	0	1	1	0
Step 5	0	0	1	0
Step 6	0	0	1	1
Step 7	0	0	0	0
Step 8	1	0	0	1

5. Hardware components

Selected Stepper Motor: Sanyo-Denki (Canon) 103H6500-7242 Stepping Motor



Fig (17) Stepper motor

motor Characteristics

Characteristic	value	unit
Holding torque at 5-phase energization	0.235 (33.28)	[N · m (oz · in) MIN.]
Rated current	0.75	A/phase
Wiring resistance	2	Ω /phase
Winding inductance	4	mH/phase
Rotor inertia	0.057 (0.31)	[$\times 10$ kg · m (oz · in)]
Mass (Weight)	0.38 (0.84)	kg (lbs)

5. Hardware components

2. Motor driver

A motor driver is a little current amplifier; the function of motor drivers is to take a low-current control signal from Arduino board (controller) and then turn it into a higher-current signal that can drive a motor, this would feed the motor with enough power and isolate it from controller.

L293D motor driver

Description

- The L293D devices are quadruple high-current half-H drivers.
- The L293D is designed to provide bidirectional drive currents of up to 600-mA
- at voltages from 4.5 V to 36 V.
- Each output is a complete totem-pole drive circuit, with a Darlington transistor sink and a pseudo-Darlington source.
- Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN.
- The L293D are characterized for operation from 0°C to 70°C.

Features

- Wide Supply-Voltage Range: 4.5 V to 36 V
- Separate Input-Logic Supply
- Internal ESD Protection
- High-Noise-Immunity Inputs
- Output Current 1 A Per Channel (600 mA for L293D)
- Peak Output Current 2 A Per Channel (1.2 A for L293D)
- Output Clamp Diodes for Inductive Transient Suppression (L293D)

5. Hardware components

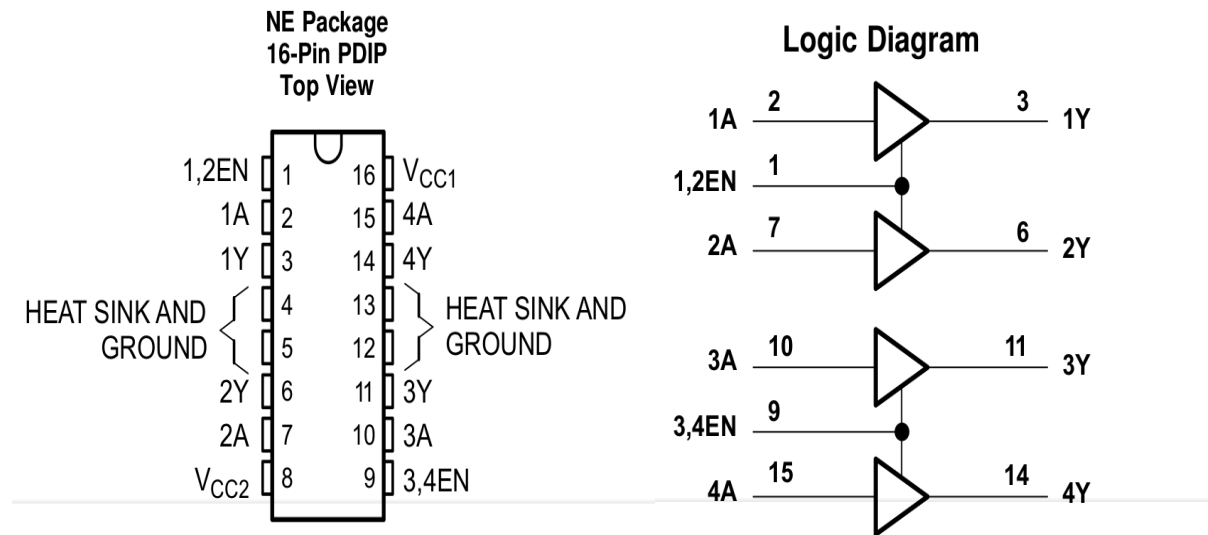


Fig (18) motor driver pin out

Pin Functions

PIN		TYPE	DESCRIPTION
NAME	NO.		
1,2EN	1	I	Enable driver channels 1 and 2 (active high input)
<1:4>A	2, 7, 10, 15	I	Driver inputs, noninverting
<1:4>Y	3, 6, 11, 14	O	Driver outputs
3,4EN	9	I	Enable driver channels 3 and 4 (active high input)
GROUND	4, 5, 12, 13	—	Device ground and heat sink pin. Connect to printed-circuit-board ground plane with multiple solid vias
V _{CC1}	16	—	5-V supply for internal logic translation
V _{CC2}	8	—	Power VCC for drivers 4.5 V to 36 V

Fig (19) Pin functions

5. Hardware components

3. Vacuum pump

Air Pump Selection:

Theory of work

mainly it's a fan powered by a 12-volt DC motor to create a partial vacuum to suck up pills. The pump will provide enough suction to lift and hold pills of various masses, ranging from 10 mg to 1.0 g.

Pump selection

The necessary force provided by the vacuum to hold pills of various masses, ranging from 10 mg to 1.0 g., therefore, is equivalent to the force needed to overcome the force of gravity on a 1.0 g pill, 0.00981 N.

$$F = ma = (1 * 10^{-3} \text{ kg}) * (9.81 \text{ m/sec}^2) = 0.00981 \text{ N}$$

first we have tried to calculate the required pump parameters values, but due to lack of information about air pumps in the market, we decided to use the try and error problem solving technique, so we tried three different air pumps from three different manufacturers until we found the "dong cheng DCXC12" vacuum cleaner (see in figure) appropriate for the task.



Fig (20) Vacuum cleaner

Vacuum cleaner description in the table below

power	12 volt DC / 3 ampere
Maximum air flow	1.2 m ³ /min)
Brand	Dong Cheng
Model Number	DCXC12

5. Hardware components

4. Pump control

The air pump will need to be turned on and off via the controller (Arduino) when a medication is needed to be dispensed. A 5-volt relay with a simple protecting circuit in figure is used to toggle the vacuum on and off



Fig (21) 5-volts relay

5. Infrared IR (safety):

we use Infrared IR to check that the air pump pulls up a pill and this pill move down to the drawer

the Infrared IR send a signal to the microcontroller if a pill move down

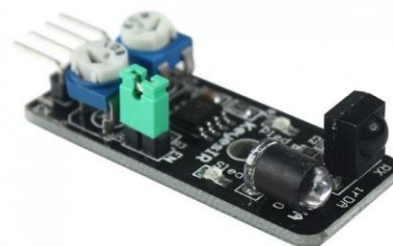
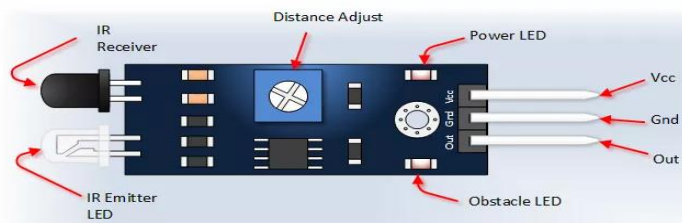


Fig (22) Infrared sensor

Specification

- Working voltage: DC 3.3V-5V
- Working current: $\geq 20\text{mA}$
- Operating temperature: $-10\text{ }^{\circ}\text{C} - +50\text{ }^{\circ}\text{C}$
- Detection distance: 2-40cm
- IO Interface: 4-wire interfaces (- / + / S / EN)
- Output signal: TTL level (low level there is an obstacle, no obstacle high)
- Adjustment: adjust multi-turn resistance
- Effective angle: 35°
- Size: $28\text{mm} \times 23\text{mm}$
- Weight Size: 9g

5. Hardware components

6. Lock “servo motor”

we use two servo motors for locking/unlocking the drawer and the warehouse door

we use a limit switch to check if the drawer closed or not when the user closes the drawer

the drawer hit the limit switch that send a signal to the microcontroller to close the drawer

by moving the servo by 90 angle.



Fig (23) Servo motor

Introduction to servo motors:

With servo motors you can position the motor shaft at a specific position (angle) using control signal. The motor shaft will hold at this position as long as the control signal not changed.

Servo motors may be classified according to size or torque that it can withstand into mini, standard and giant servos. Usually mini and standard size servo motors can be powered by Arduino directly with no need to external power supply or driver.

Usually servo motors come with arms (metals or plastic) that is connected to the object required to move.

How does servo works?

Servo has 3 wires (Black: ground, Red: +5, Colored: control signal)

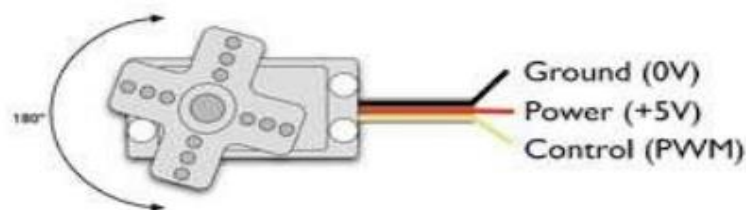


Fig (24) Servo motor pinout

5. Hardware components

The third pin accept the control signal which is a pulse-width modulation (PWM) signal. It can be easily produced by all micro- controllers and Arduino board. This accepts the signal from your controller that tells it what angle to turn to. The control signal is fairly simple compared to that of a stepper motor. It is just a pulse of varying lengths. The length of the pulse corresponds to the angle the motor turns to.

The pulse width sent to servo ranges as follows:

Minimum: 1 millisecond ---> Corresponds to 0 rotation angle.

Maximum: 2 milliseconds ---> Corresponds to 180 rotation angle.

Any length of pulse in between will rotate the servo shaft to its corresponding angle. For example, 1.5 MS pulse corresponds to rotation angle of 90 degrees.

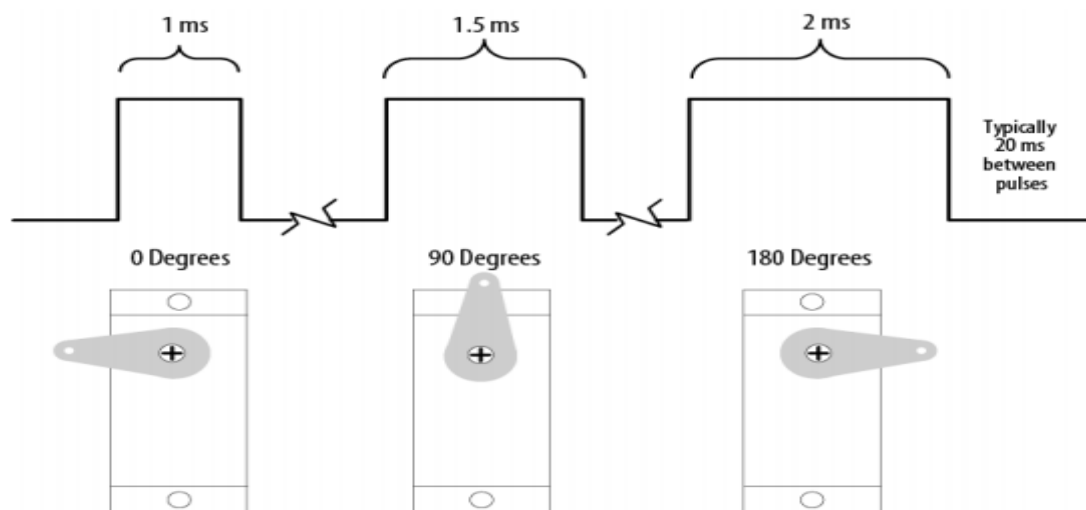


Fig (25) Servo motor motion

5. Hardware components

Motor control using Arduino: As Arduino software comes with a sample servo sketch and servo library that will get you up and running quickly. Just connect servo wires to Arduino as shown in figure and upload the code to test it.

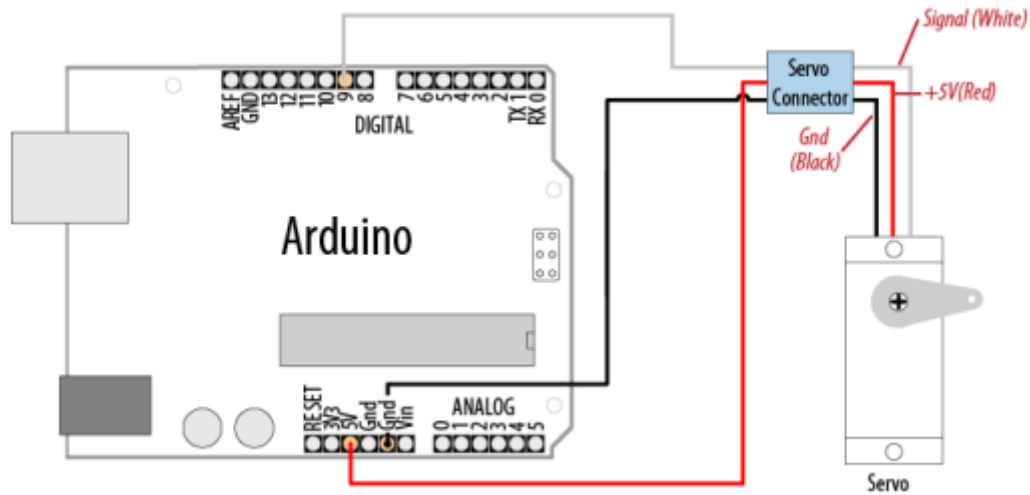


Fig (26) Servo Motor-Arduino connection

1. Connect the black wire from the servo to the Gnd pin on the Arduino
2. Connect the red wire from the servo to the +5V pin on the Arduino
3. Connect the third wire (usually orange or yellow) from the servo to a digital pin on the Arduino

5. Hardware components

7. Motors initialization

- a) **Tray initialization:** after the user put the medicine in the warehouse it must that the tray moves the position where that hollow warehouse down the nozzle on the move up down motor to do this we used an infrared IR. The tray will move and when the barrier pass in front of the infrared IR the infrared IR will send signal to the microcontroller to stop the stepper motor

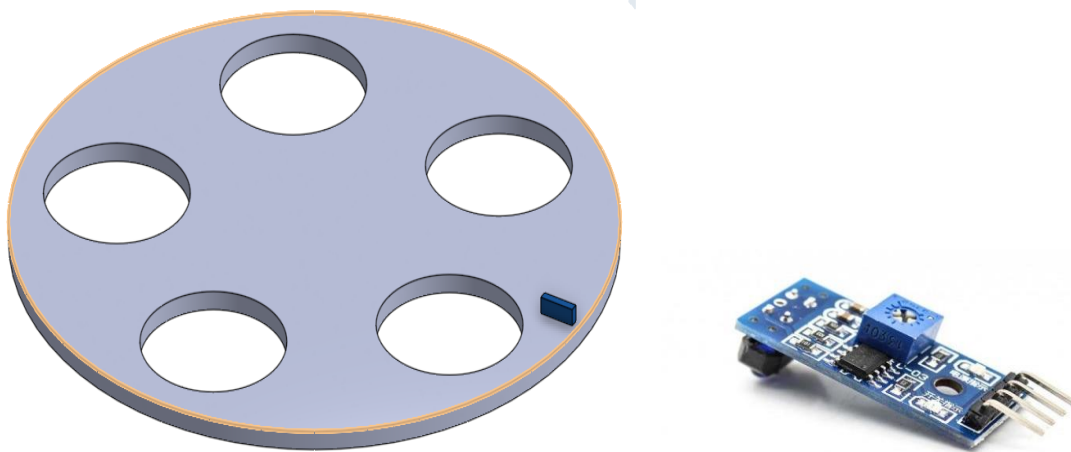


Fig (27) Tray and Infrared sensor

- b) **The nozzle apparatus:** To initial the position of the nozzle apparatus we use a limit switch that put in a side of the box and when the nozzle hit it the limit switch will send signal to the microcontroller to stop the stepper motor



Fig (28) Limit switch

5. Hardware components

8. Power supply

the project components need five and twelve voltages with maximum total current draw of 8 amperes. table () contains the power rating values for each component.

the tray [stepper motor]	5 DC volt @ .75 A
the nozzle apparatus [stepper motor]	5 DC volt @ .75 A
locker lock servo motor	5 DC volt
warehouse door lock servo motor	5 DC volt
Arduino Uno	5 DC volt @ $I < 1A$
raspberry Pi 3	5 DC volt @ 2.5A
air pump	12 Dc volt @ 3A

So we did use A power supply unit that converts from AC to different low-voltage regulated DC power (3.3,5,12, -12).



Fig (29) power supply

5. Hardware components

9. Controller (Arduino Uno)

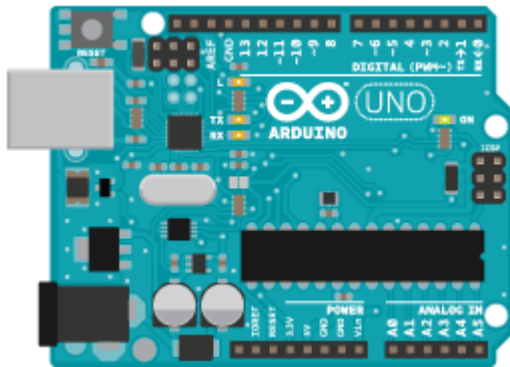


Fig (30) Arduino

The Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button.

It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

Arduino connects with the raspberry by USB and receive the command by using pySerial python library.

Main Tasks:

- Dispense pills
- Open/Secure warehouse
- Open/Secure drawer
- Send sensors' readings

6. Software Introduction

The purpose for this part of project is to provide a way for care giver/ Senior citizen to insert the schedule, and send the serial commands to Arduino at the specific time of schedule to dispense the medications.

Also it's desired to provide some more features like Location and Vital signs tracking, a solution for emergency cases and medical profile, Also box should be able to dispense schedules even if it's offline.

we achieved to do so using:

- Raspberry-pi: to get schedule from user over the internet and save it for offline use
- Android-application: to add schedules, send/receive emergency notifications and senior citizen tracking

Actually the app is two separate apps altogether; one for senior citizen and one for care giver combined together.

Type of app is chosen during sign-in process according to user selection.

Both apps have some similar functions but each one has some unique properties such as

- senior citizen app can send emergency, but care giver's app can only receive it.
- Location/ Vital signs tracking services are embedded in senior citizen's app not care giver's and so on

Note: all functions are in the same app but according to app type user selected during sign-in process functions are activated

Main Features:

- Multi user
- add/delete schedule
- Add pills to box
- Box Should work offline after adding the schedule
- Dispense schedule
- Track pill count in the box
- Send commands to the box
- Display status to user about the box
- Send notification to both senior citizen and care giver
- Track vital signs
- Location tracking
- Fast way for emergency
- Medical profile
- IOT application "proof of concept"

7. Connection

1. Flow chart

a) Schedule

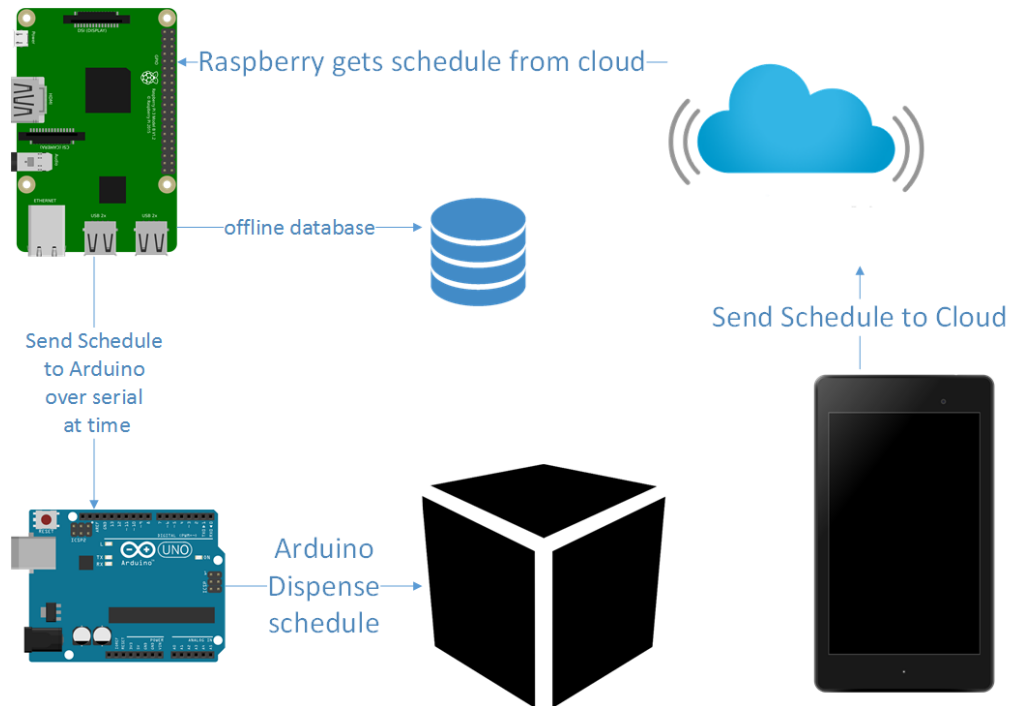


Fig (31) Schedule flowchart

b) Emergency

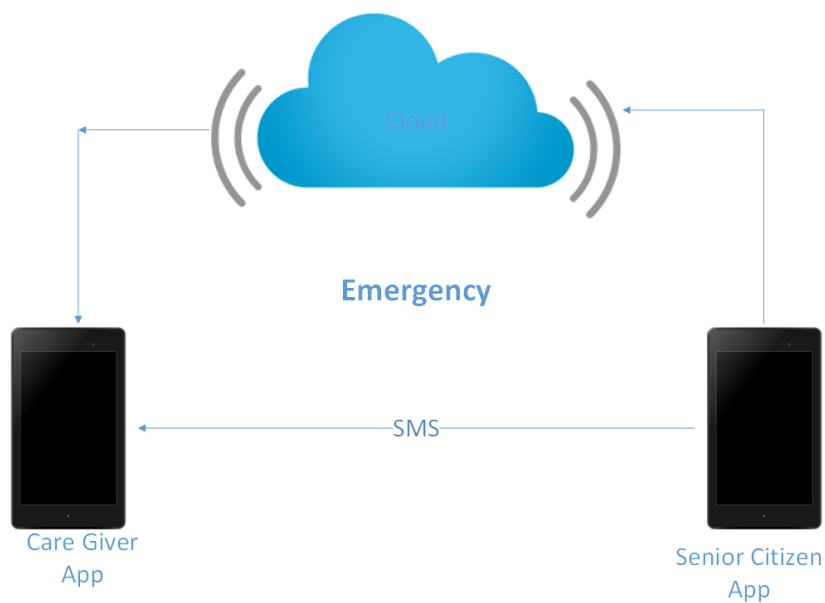


Fig (32) Emergency flowchart

c) Smart Band

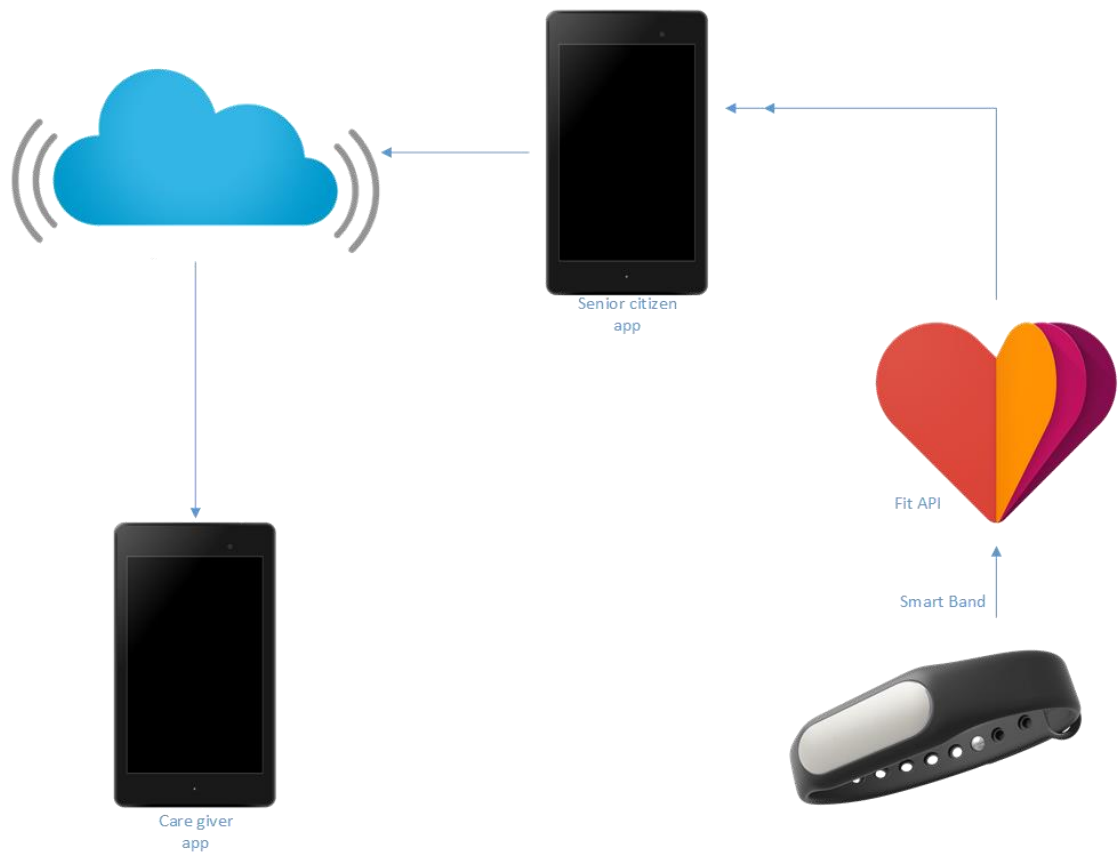


Fig (33) Smart band connection flowchart

2. Rejected connection schemas

a) Raspberry as a server

- requires static IP, port forward which is hard to implement,
- also a headache Setup for users,
- and hard task for Raspberry

b) database sync:

- Requires Raspberry-pi to be a server,
- not a good decision as users will have an Editor account for the database which is a very risky way to implement the project,
- not a proper way for mobile devices as they will pull the server, every X seconds which will drain battery,
- also not a real time solution.

c) API:

- Requires Raspberry-pi to be a server,
- A better solution as no account is needed "only API-Key",
- less data can be sent and received as queries can be adjusted,
- and queries can be checked before passing it to database,
- but requires a full change of queries which was already implemented.

d) PHP website:

- Requires Raspberry-pi to be a server,
- hard to sync between PHP and python script.

e) python website:

- Requires Raspberry-pi to be a server.

f) XMPP, MQTT:

- Requires Raspberry-pi to be a server,
- Require Android Always-on Connection which isn't a good decision for batteries.

3. Firebase

Why Firebase?

Firebase Cloud messaging

FCM uses the already opened connection with google servers, also FCM is better real time implementation as it can wake up device in case of important notifications even if the app is not working the mobile receives the messages and open the app to decide what to do with it.

Google NoSQL database:

Till now is the best implementation as the connection is more reliable, no need to open external connection, secured database with JWT OAuth2, And the database is real time using http-SSE stream server event

What is Firebase platform?

Firebase is a mobile and web application development platform. Firebase is made up of complementary features that developers can mix-and-match to fit their needs.

Firebase helps you develop high-quality apps, grow your user base, and earn more money. Each feature works independently, and they work even better together.

Trusted by the largest Development teams around the world—including NPR, Shazam, Duolingo, The newyork times, Alibaba, trivago and Venmo.

Main Features:

- a) **Real-time Database:** Store and sync data between users and devices in real-time using a cloud-hosted, NoSQL database. Updated data syncs across connected devices in milliseconds, and data remains available if your app goes offline, providing a great user experience regardless of network connectivity.
- b) **Authentication:** Manage your users in a simple and secure way. Firebase Auth offers multiple methods to authenticate, including email/password, third-party providers like Google or Facebook, or using your existing account system directly. Build your own interface, or take advantage of our open source, fully customizable UI.
- c) **Cloud Functions:** Extend your app with custom backend code without needing to manage and scale your own servers. Functions can be triggered by events, which are emitted by Firebase products, Google Cloud services, or third parties, using web hooks.
- d) **Cloud Messaging:** Send messages and notifications to users across platforms—Android, iOS, and the web—for free. Messages can be sent to single devices, groups of devices, or specific topics or user segments. FCM scales to even the largest apps, delivering hundreds of billions of messages per day.

4. Firebase Components description

I. **Firebase Real-time Database:**

Store and sync data with NoSQL cloud database. Data is synced across all clients in real-time, and remains available when app goes offline.

The Firebase Real-time Database is a cloud-hosted database. Data is stored as JSON and synchronized in real-time to every connected client.

all of your clients share one Real-time Database instance and automatically receive updates with the newest data.

Key capabilities

- **Real-time:** Instead of typical HTTP requests, the Firebase Real-time Database uses data synchronization—every time data changes, any connected device receives that update within milliseconds. Provide collaborative and immersive experiences without thinking about networking code.
- **Offline:** Firebase apps remain responsive even when offline because the Firebase Real-time Database SDK persists your data to disk. Once connectivity is reestablished, the client device receives any changes it missed, synchronizing it with the current server state.
- **Accessible from Client Devices:** The Firebase Real-time Database can be accessed directly from a mobile device or web browser; there's no need for an application server. Security and data validation are available through the Firebase Real-time Database Security Rules, expression-based rules that are executed when data is read or written.

How does it work?

The Firebase Real-time Database lets you build rich, collaborative applications by allowing secure access to the database directly from client-side code. Data is persisted locally, and even while offline, real-time events continue to fire, giving the end user a responsive experience. When the device regains connection, the Real-time Database synchronizes the local data changes with the remote updates that occurred while the client was offline, merging any conflicts automatically.

7. Connection

The Real-time Database provides a flexible, expression-based rules language, called Firebase Real-time Database Security Rules, to define how your data should be structured and when data can be read from or written to. When integrated with Firebase Authentication, developers can define who has access to what data, and how they can access it.

The Real-time Database is a NoSQL database and as such has different optimizations and functionality compared to a relational database. The Real-time Database API is designed to only allow operations that can be executed quickly. This enables you to build a great real-time experience that can serve millions of users without compromising on responsiveness. Because of this, it is important to think about how users need to access your data and then structure it accordingly.

Implementation path

- **Integrate the Firebase Real-time Database SDKs:** Quickly include clients via Gradle, CocoaPods, or a script include.
- **Create Real-time Database References:** Reference your JSON data, such as "users/user:1234/phone_number" to set data or subscribe to data changes.
- **Set Data and Listen for Changes:** Use these references to write data or subscribe to changes.
- **Enable Offline Persistence:** Allow data to be written to the device's local disk so it can be available while offline.
- **Secure your data:** Use Firebase Real-time Database Security Rules to secure your data.

To store other types of data?

Firebase Remote Config stores developer specified key-value pairs to change the behavior and appearance of your app without requiring users to download an update.

Firebase Hosting hosts the HTML, CSS, and JavaScript for your website as well as other developer-provided assets like graphics, fonts, and icons.

Cloud Storage stores files such as images, videos, and audio as well as other user-generated content.

II. Firebase Authentication

Most apps need to know the identity of a user.

Knowing a user's identity allows an app to securely save user data in the cloud and provide the same personalized experience across all of the user's devices.

Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to your app.

It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more.

Firebase Authentication integrates tightly with other Firebase services, and it leverages industry standards like OAuth 2.0 and OpenID Connect, so it can be easily integrated with your custom backend.

Firebase SDK Authentication

- **Email and password based authentication:** Authenticate users with their email addresses and passwords. The Firebase Authentication SDK provides methods to create and manage users that use their email addresses and passwords to sign in. Firebase Authentication also handles sending password reset emails,
- **Phone number authentication,**
- **Custom auth system integration,**
- **Anonymous auth,**
- **Federated identity provider integration:** allow users to sign in with their Google, Facebook, Twitter, and GitHub accounts.

How does it work?

- To sign a user into your app, you first get authentication credentials from the user.
- These credentials can be the user's email address and password, or an OAuth token from a federated identity provider.
- Then, you pass these credentials to the Firebase Authentication SDK.
- Our backend services will then verify those credentials and return a response to the client.
- After a successful sign in, you can access the user's basic profile information, and you can control the user's access to data stored in other Firebase products.
- You can also use the provided authentication token to verify the identity of users in your own backend services.

Implementation paths

Using the Firebase Authentication SDK

- Set up sign-in methods For email address and password or phone number sign-in and any federated identity providers you want to support, enable them in the Firebase console and complete any configuration required by the identity provider, such as setting your OAuth redirect URL.
- Implement UI flows for your sign-in methods For email address and password sign-in, implement a flow that prompts users to type their email addresses and passwords. For phone number sign-in, create a flow that prompts users for their phone number, and then for the code from the SMS message they receive. For federated sign-in, implement the flow required by each provider.
- Pass the user's credentials to the Firebase Authentication SDK
Pass the user's email address and password or the OAuth token that was acquired from the federated identity provider to the Firebase Authentication SDK.

III. Firebase Cloud Messaging

Firebase Cloud Messaging (FCM) is a cross-platform messaging solution that lets you reliably deliver messages at no cost.

Using FCM, you can notify a client app that new email or other data is available to sync.

You can send notification messages to drive user reengagement and retention.

For use cases such as instant messaging, a message can transfer a payload of up to 4KB to a client app.

Key capabilities

- **Send notification messages or data messages:** Send notification messages that are displayed to your user. Or send data messages and determine completely what happens in your application code.
- **Versatile message targeting:** Distribute messages to your client app in any of three ways — to single devices, to groups of devices, or to devices subscribed to topics.
- **Send messages from client apps:** Send acknowledgments, chats, and other messages from devices back to your server over FCM's reliable and battery-efficient connection channel.

How does it work?

An FCM implementation includes two main components for sending and receiving:

1. A trusted environment such as Cloud Functions for Firebase or an app server on which to build, target and send messages.
2. An iOS, Android, or Web (JavaScript) client app that receives messages.

Message types

With FCM, you can send two types of messages to clients:

1. **Notification messages:** sometimes thought of as "display messages."
2. **Data messages:** which are handled by the client app.

You can send messages via the Admin SDK or the HTTP and XMPP APIs.

For testing or for sending marketing or engagement messages with powerful built-in targeting and analytics, you can also use the Notifications composer.

7. Connection

Implementation path

- Set up the FCM SDK: Set up Firebase and FCM on your app according the setup instructions for your platform.
- Develop your client app: Add message handling, topic subscription logic, or other optional features to your client app. During the development, you can easily send test messages from the Notifications composer.
- Develop your app server: Decide whether you want to use the Admin SDK or one of the server protocols to create your sending logic — logic to authenticate, build send requests, handle responses, and so on. Then build out the logic in your trusted environment. Note that if you want to use upstream messaging from your client applications, you must use XMPP, and that Cloud Functions does not support the persistent connection required by XMPP.

IV. Cloud Functions for Firebase

Cloud Functions for Firebase lets you automatically run backend code in response to events triggered by Firebase features and HTTPS requests. Your code is stored in Google's cloud and runs in a managed environment. There's no need to manage and scale your own servers.

Key capabilities

The functions you write can respond to events generated by these other Firebase and Google Cloud features:

- Real-time Database Triggers
- Firebase Authentication Triggers
- Google Analytics for Firebase Triggers
- Cloud Storage Triggers
- Cloud Pub/Sub Triggers
- HTTP Triggers

How does it work?

After you write and deploy a function, Google's servers begin to manage the function immediately, listening for events and running the function when it is triggered.

As the load increases or decreases, Google responds by rapidly scaling the number of virtual server instances needed to run your function.

Lifecycle of a function

- The developer writes code for a new function, selecting an event provider, and defining the conditions under which the function should execute.
- The developer deploys the function, and Firebase connects it to the selected event provider.
- When the event provider generates an event that matches the function's conditions, the code is invoked.
- If the function is busy handling many events, Google creates more instances to handle work faster. If the function is idle, instances are cleaned up.
- When the developer updates the function by deploying updated code, all instances for the old version are cleaned up and replaced by new instances.

7. Connection

Implementation path

- **Set up Cloud Functions:** Install the Firebase CLI and initialize Cloud Functions in your Firebase project.
- **Write functions:** Write JavaScript code to handle events from Firebase services, Google Cloud services, or other event providers.
- **Deploy and monitor:** Deploy your functions using the Firebase CLI. You can use the Firebase console to view and search through your logs.

5. Google FIT

Google Fit is an open ecosystem that allows developers to upload fitness data to a central repository where users can access their data from different devices and apps in one location:

Fitness apps can store data from any wearable or sensor.

Fitness apps can access data created by any app.

User's fitness data is persisted when they upgrade their fitness devices.

Components

- **The fitness store:** A central repository that stores data from a variety of devices and apps. The fitness store is a cloud service that is transparent to clients.
- **The sensor framework:** A set of high-level representations that make it easy to work with the fitness store. You use these representations with the Google Fit APIs.
- **Permissions and user controls:** A set of authorization scopes to request user permission to work with fitness data. Google Fit requires user consent to access fitness data.

Google Fit APIs

- **The sensor framework:** The sensor framework defines high-level representations for sensors, fitness data types, data points, and sessions. These representations make it easy to work with the fitness store on any platform.
- **Data Sources:** Data sources represent sensors and consist of a name, the type of data collected, and other sensor details. A data source may represent a hardware sensor or a software sensor. You can define software sensors in your apps.
- **Data Types:** Data types represent different kinds of fitness data, like step count or heart rate. Data types establish a schema through which different apps can understand each other's data. A data type consists of a name and an ordered list of fields, where each field represents a dimension. For example, a data type for location contains three fields (latitude, longitude, and accuracy), whereas a data type for weight contains only one field.

7. Connection

- **Data Points:** Data points consist of a timestamped array of values for a data type, read from a data source. You use data points to record and insert fitness data in the fitness store, and to read raw data from a data source. Points that contain a start time represent a time range instead of an instantaneous reading.
- **Datasets:** Datasets represent a set of data points of the same type from a particular data source covering some time interval. You use datasets to insert data into the fitness store. Queries to read data from the fitness store also return datasets.
- **Sessions:** Sessions represent a time interval during which users perform a fitness activity, such as a run, a bike ride, and so on. Sessions help organize data and perform detailed or aggregate queries on the fitness store for a fitness activity.

Permissions and user controls

Google Fit requires user consent before apps can read or store fitness data. Google Fit defines OAuth scopes that map to three permission groups with separate read and write privileges: activity, location, and body. Each permission group grants apps access to a set of data types. Apps specify one or more of these scopes to work with fitness data, and Google Fit requests the corresponding permissions from the user.

Google Fit provides the following APIs:

- Android APIs for Android apps.
- REST API for apps on any platform.

1. Introduction

Why Raspberry Pi?

A Raspberry Pi is a credit card-sized computer originally designed for education, inspired by the 1981 BBC Micro.

Creator Eben Upton's goal was to create a low-cost device that would improve programming skills and hardware understanding at the pre-university level.

But thanks to its small size and accessible price, it was quickly adopted by tinkerers, makers, and electronics enthusiasts for projects that require more than a basic microcontroller (such as Arduino devices).

The Raspberry Pi is slower than a modern laptop or desktop but is still a complete Linux computer and can provide all the expected abilities that implies, at a low-power consumption level.

as a Linux machine a lot of high level features can be added using high level languages like database connection

Why Python?

Python is a widely used high-level programming language for general-purpose programming. An interpreted language,

Python has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords),

and a syntax which allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java.

The language provides constructs intended to enable writing clear programs on both a small and large scale.

although python is scripting language it's much stronger than many languages especially on raspberry pi

due to its high level abstraction and libraries

so almost every basic task like parse Jason data API is available by only one line of code

and it's also the official language of raspberry pi so more resources are available on the internet

8. Box client

Why Python 2.7?

unfortunately, python 3 isn't back-compatible so python2 scripts will not work on python3 interpreter "in most cases"

python2 has more resources on the internet, also project's script depends on libraries that is available only on python2

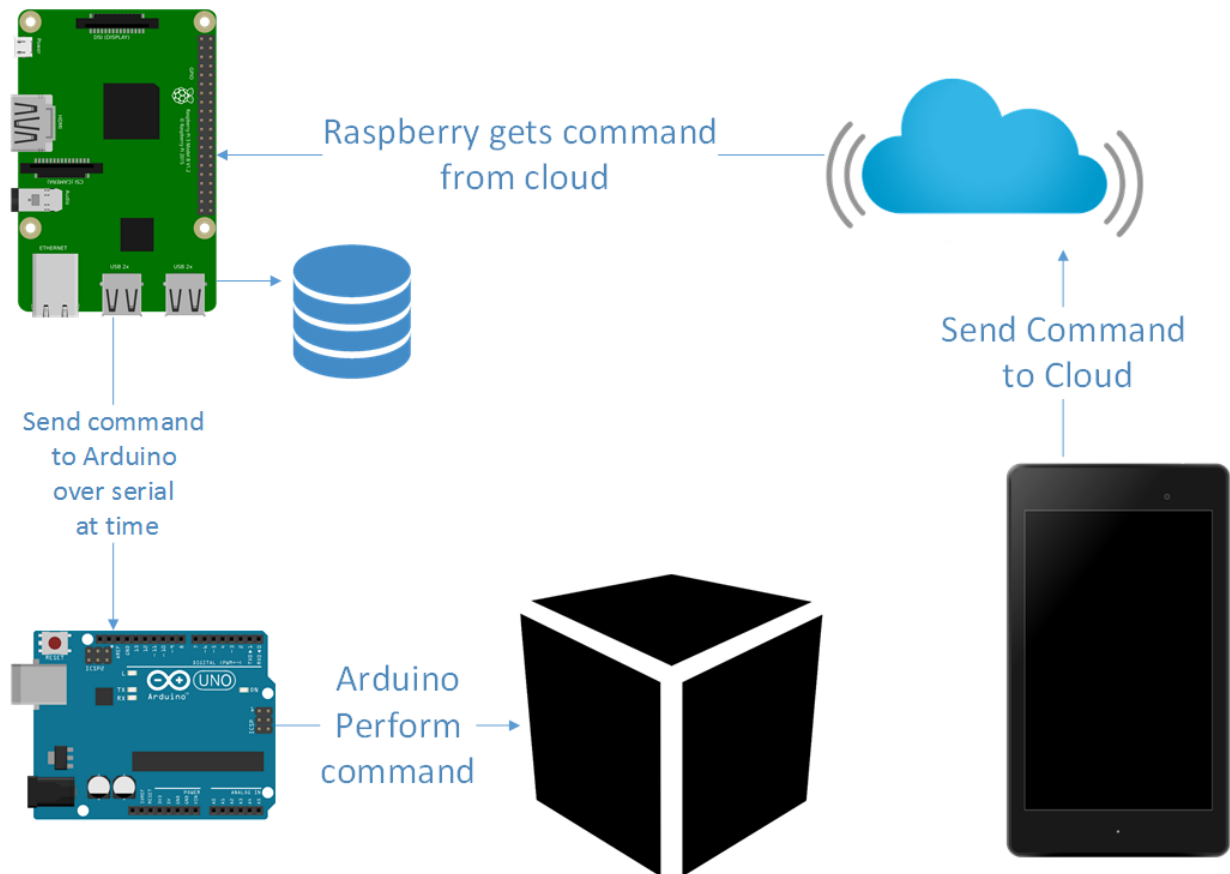


Fig (34) Mobile-Box connection

2. Raspberry-Pi

Raspberry-pi main task is to save schedule offline, dispense pills at time, and provide high level features to the box like connecting to the internet to access the cloud

to be able to send/receive data from mobile apps via cloud
data includes schedule, pill count and commands for the box.

it sends commands to Arduino using serial communication; pySerial python library is used for this purpose

it provides an easy way for serial communication between python and any platform.

Main Features:

- Connect to mobile apps via cloud
- Send/Receive data at real time
- Save schedule offline
- Connect to Arduino via serial channel
- Send commands to Arduino
- Dispense pills at time
- Track pill count
- Send notifications to both android apps
- Small IOT application

1. Introduction

Why Android?

Android is a mobile operating system developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets.

Android's user interface is mainly based on direct manipulation, using touch gestures that loosely correspond to real-world actions, such as swiping, tapping and pinching, to manipulate on-screen objects, along with a virtual keyboard for text input.

In addition to touchscreen devices, Google has further developed Android TV for televisions, Android Auto for cars, and Android Wear for wrist watches, each with a specialized user interface. Variants of Android are also used on notebooks, game consoles, digital cameras, and other electronics.

Android's source code is released by Google under an open source license.

easy to learn and easy to code

also google provides high level APIs to help developers reach their goal both easily and effectively

Why Android Studio?

is the official integrated development environment (IDE) for the Android platform.

It was announced on May 16, 2013 at the Google I/O conference.

Android Studio was in early access preview stage starting from version 0.1 in May 2013, then entered beta stage starting from version 0.8 which was released in June 2014. The first stable build was released in December 2014, starting from version 1.0

Based on JetBrains' IntelliJ IDEA software, Android Studio is designed specifically for Android development.

Why java?

while there are many other languages officially supported like GO, C++, kotlin java is the native language for android applications and is officially supported also has an enormous libraries and APIs, and much more resources on the internet

2. Android Application

Android app main task is to provide users with an easy way to add/edit schedule, send/receive notifications/emergency, and senior citizen vital-signs/location tracking.

android app has three main components

- Activities/Fragments
- Services
- Broadcast receivers

Activities/fragments: displays info to the user when user opens the app

An Activity is an application component that provides a screen, with which users can interact in order to do something.

Whereas a Fragment represents a behavior or a portion of user interface in an Activity that can be re-used in multiple activities.

Services: code portions that must be running all the time whether app is open or not, runs automatically in background.

used to track location, track vital signs and detect shake event to send emergency.

Broadcast receivers: code portion that will run when certain event occurs like "boot_compelete" to automatically run services, or when "sms_received" to send emergency notification to care giver if it's emergency SMS from senior citizen.

Note: google recommendations are followed to provide better user experience.

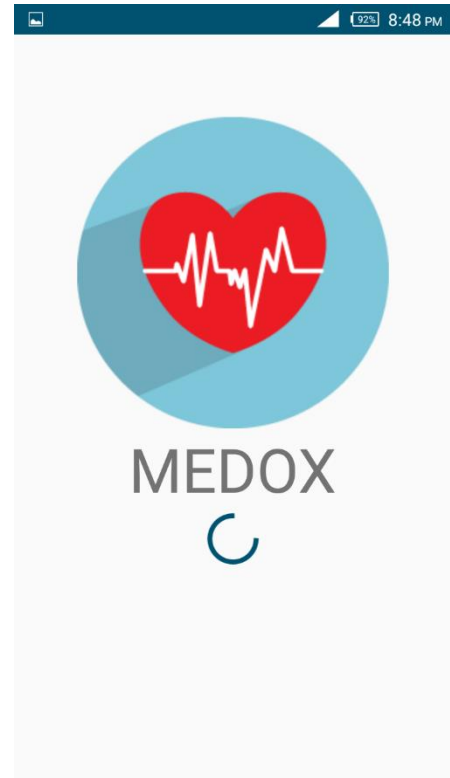
3. Android app features include:

- Real time communication
- Authenticate users
- Choose user type "senior citizen/ care giver"
- Track vital signs
- Track location
- Detect shake event
- Send/Receive notification/emergency
- Fast way for emergency
- Display current status
- Add/Edit schedule
- Add/Edit pill count in the warehouse
- Send commands to the box
- Medical profile
- Can work offline to view data
- Can add data offline to be sent when connection is restored

4. Android Screens

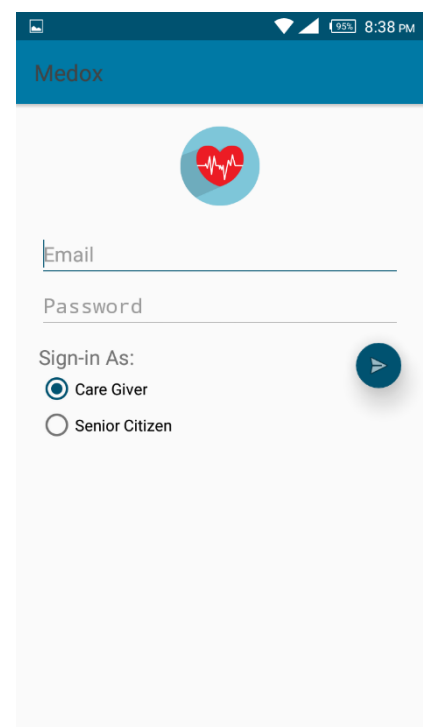
1. Splash Activity

- make sure user is authenticated; otherwise, redirect to authentication activity
- redirect user to home activity according to app type
- makes sure that services are running if app type is senior citizen
- request required permissions for app from user like SMS and location "for android 6+"
- check if user is configured; otherwise redirect to settings activity



2. Authentication Activity

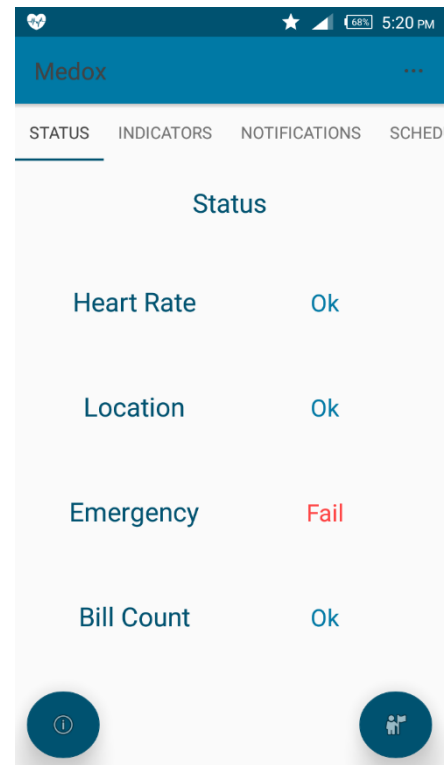
- responsible for getting email and password from user for authentication
- and gets user app type according to user choice
- if user is authenticated redirect to splash screen to start services and redirect to home activity according to app type



9. Mobile client

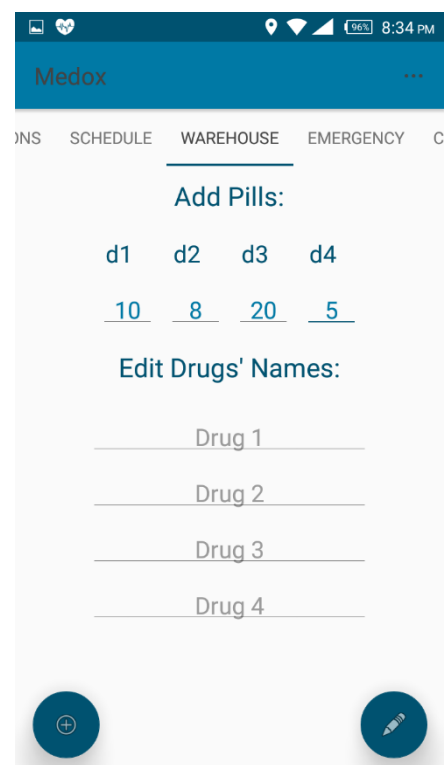
3. status screen

- displays a status of the box and senior citizen's trackers.
- if there is any problem care giver has a fast way to check it
- buttons redirect to notification and emergency screens respectively



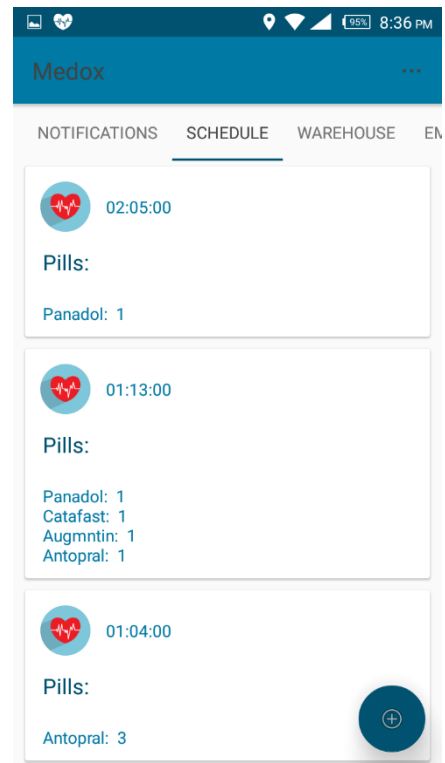
4. warehouse screen

- provides a way for user to add pills to pill count of the box
- also a way to set drugs names' so warehouses numbers doesn't need to be remembered



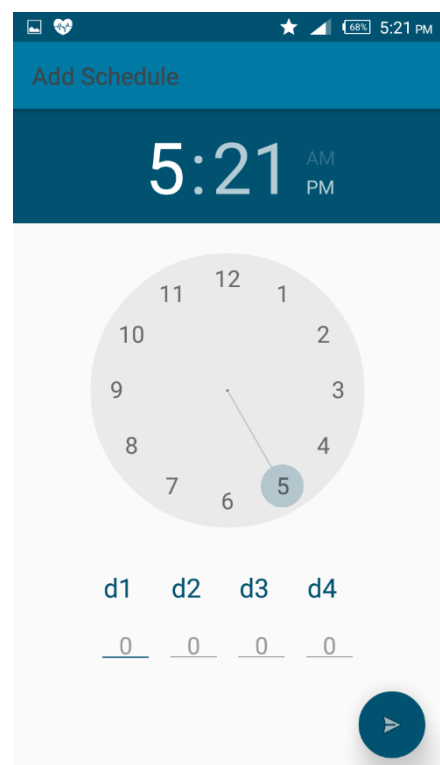
5. schedule screen

- displays current saved schedule from database
- schedule can be removed by simply long press on it
- and to add a new schedule, simply press add button



6. add schedule Activity

- an easy way for user to add new schedule, just by setting desired time and how many pills from each medication
- and send that to database, which Raspberry-pi will detect the change and updates its timetable and offline database

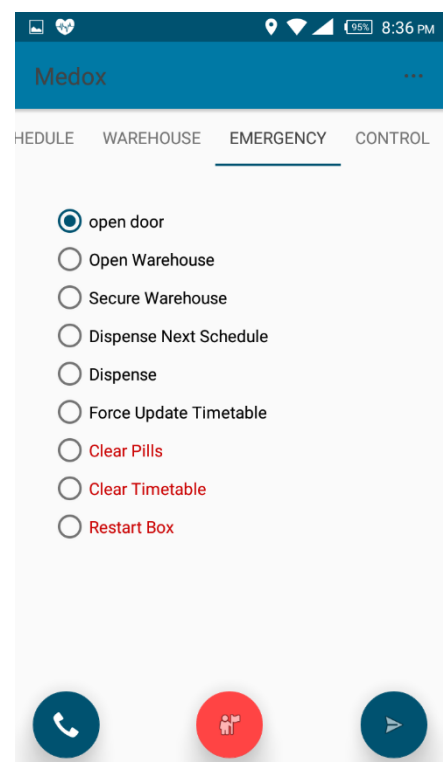
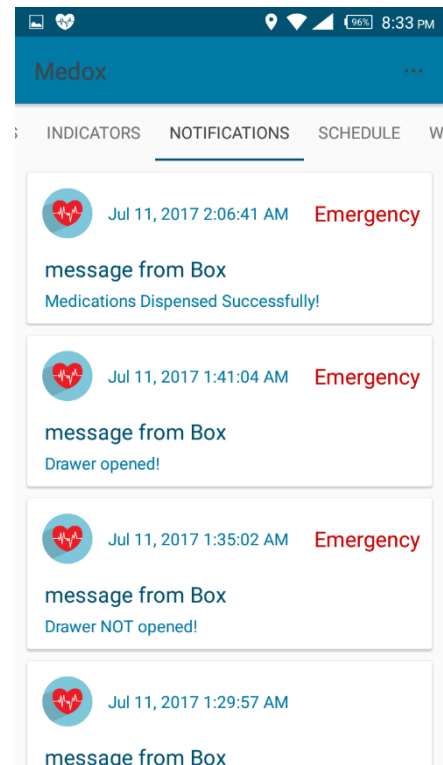


7. notification screen

- displays current and old notifications to user which is saved in database ordered by time

8. emergency screen

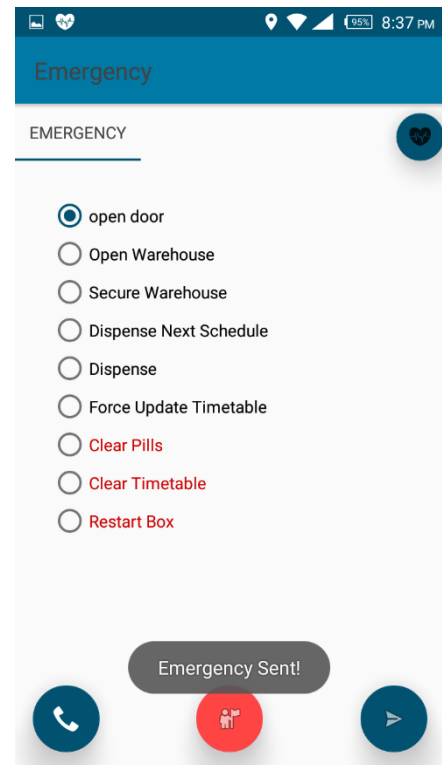
- screen for emergency tasks, also provides control functions of the box
- tasks include
 - open box door lock
 - open box warehouse lock
 - secure box warehouse lock
 - dispense next schedule now
 - dispense any combination of bills now "without affecting schedule"
 - force Raspberry-pi to update its offline database
 - reset pill count in the box
 - delete all timetable entries
 - restart the box
 - call senior citizen/care giver according to app-type
 - send emergency to care giver
 - emergency button appears only if app type is senior citizen
 - emergency sent over the internet and also as SMS



9. Mobile client

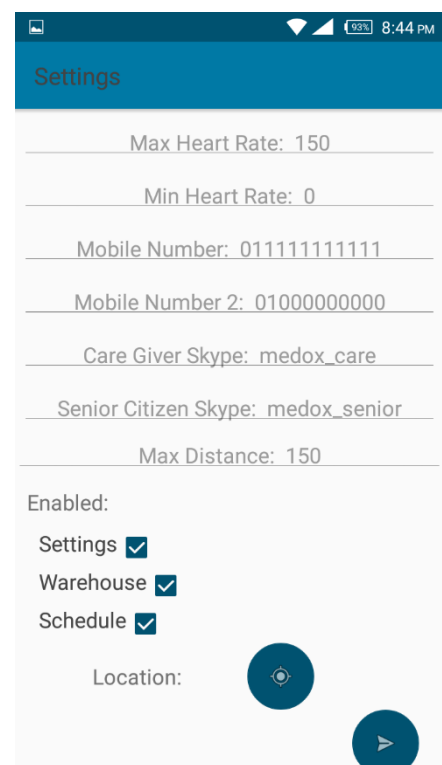
9. Emergency Activity

- starts in any emergency situations like under/over limit heart rate, senior citizen out of safe distance from home or shake event
- the main purpose of it is that it overrides lock screen so gives a faster way to perform emergency tasks



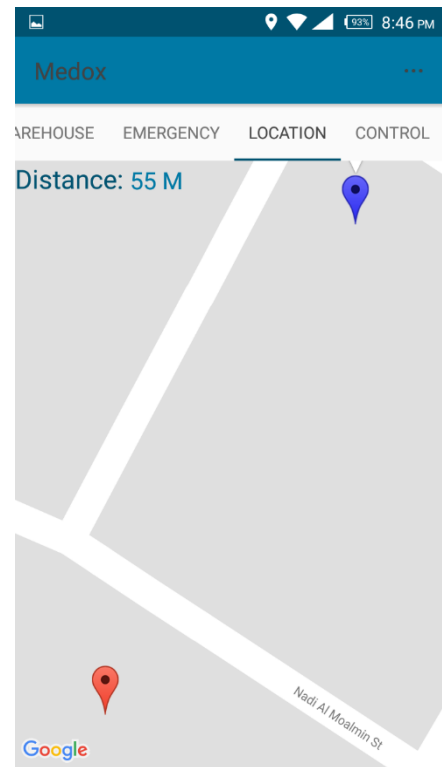
10. Settings Activity

- configuration screen for user to set different app settings
- settings include:
 - max/min heart rate safe limits
 - emergency phone numbers "to send SMS to"
 - care giver and senior citizen skype names to perform call
 - home location
 - safe distance from home
 - enabled "tasks senior citizen can perform"



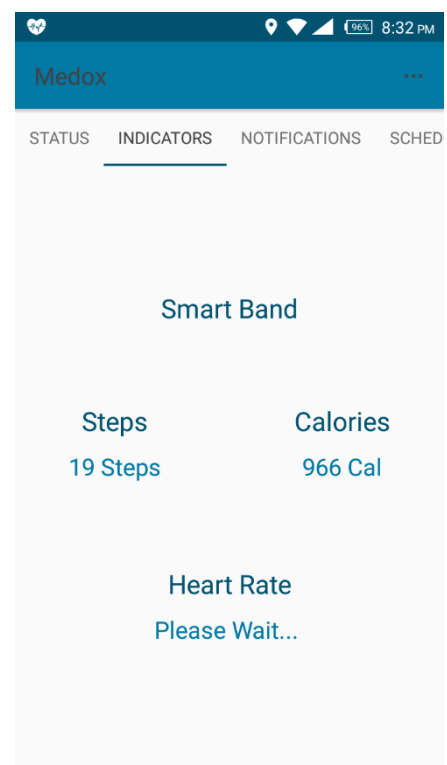
11. location screen

- only appears in care giver's app
- used to track senior citizen location in real time



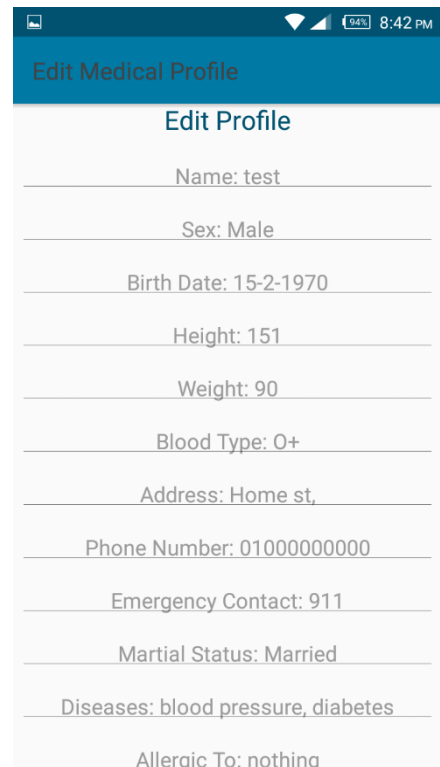
12. indicators screen

- a simple display of senior citizen measured vital signs
- which are stored and queried in real-time



13. edit medical profile activity

- used to set or edit medical profile of senior citizen
- only available in care giver's app
- profile includes several fields as shown in the picture

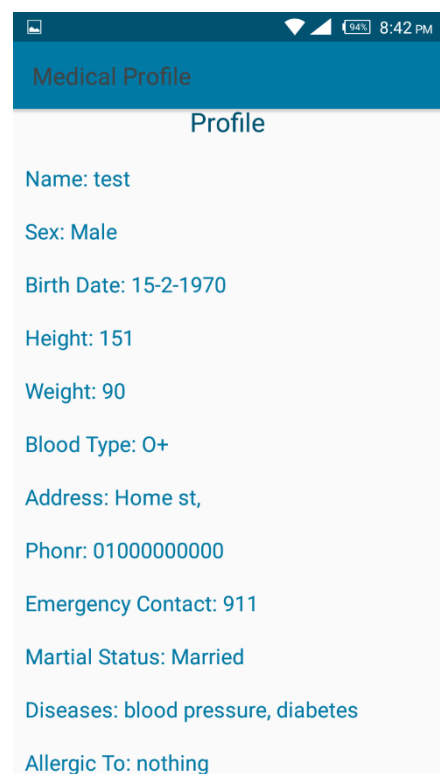


The screenshot shows the 'Edit Medical Profile' activity. The title bar is blue with the text 'Edit Medical Profile'. Below the title bar, the text 'Edit Profile' is centered. The form contains the following fields:

Name: test
Sex: Male
Birth Date: 15-2-1970
Height: 151
Weight: 90
Blood Type: O+
Address: Home st,
Phone Number: 01000000000
Emergency Contact: 911
Martial Status: Married
Diseases: blood pressure, diabetes
Allergic To: nothing

14. profile activity

- simple activity to display senior citizen medical information

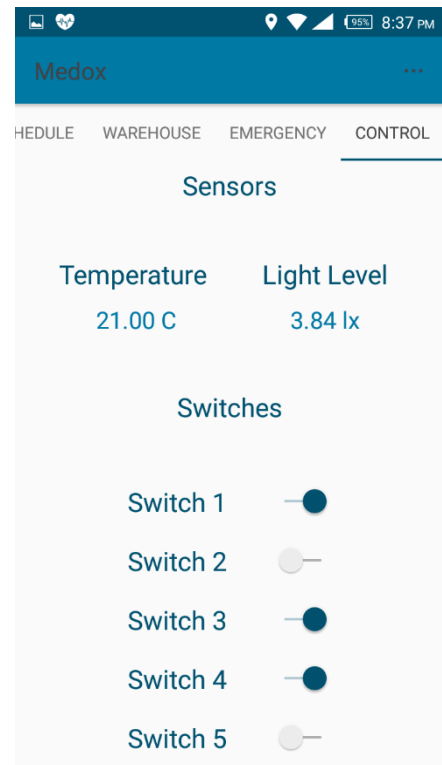


The screenshot shows the 'Medical Profile' activity. The title bar is blue with the text 'Medical Profile'. Below the title bar, the text 'Profile' is centered. The form displays the following information:

Name: test
Sex: Male
Birth Date: 15-2-1970
Height: 151
Weight: 90
Blood Type: O+
Address: Home st,
Phonr: 01000000000
Emergency Contact: 911
Martial Status: Married
Diseases: blood pressure, diabetes
Allergic To: nothing

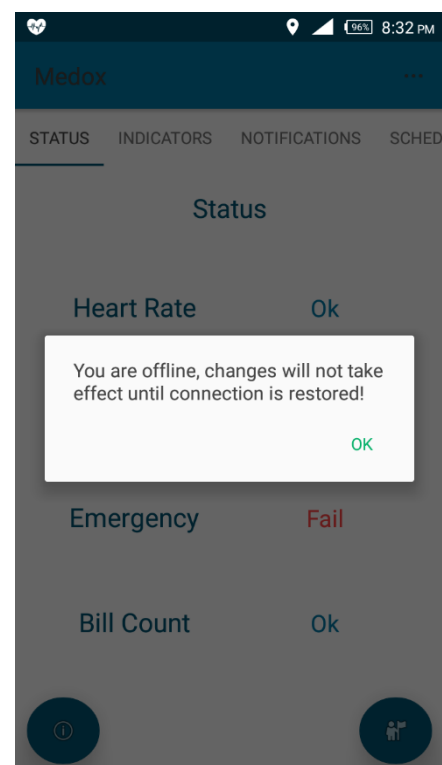
15. control screen "proof of concept"

- provides real-time readings, control from the box
- displays sensors' readings from the box
- controls 8- switches connected in the box



16. offline pop-up

- pop-up if application is not connected to the internet
- to inform user that application is offline but changes will be saved and will take effect when connection restored



1. Solved problems

- **Connection problem:**
 - the problem was to find a way for senior, care apps and raspberry to communicate with each other
 - and a place to store data for users
 - **solution:** firebase platform is used
- **Thread sync:**
 - **Solution:** using module contains Boolean variables all threads can access for reading or updating values
- **Android java Null pointer for objects:**
 - by default, any android app will crash if a null pointer exception happens at any time during run time
 - this can happen if app tried to get value from database which isn't there yet
 - **solution:** adding extra null objects check
 - Note: Kotlin solved java null pointer exceptions by default
 - but kotlin not used in project as it was introduced later when the app was already up and running
- **Time variables:**
 - time is saved in database as string, so to calculate the remaining time to next schedule
 - it was important to convert it to convert it to some sort of time variable on python
 - the problem is python can't handle converting it
 - **Solution:** a conversion function was developed to solve this problem
 - by splitting the string by ":", convert hours, minutes, seconds to integers
 - then create a new time variable and calculate the delay in seconds

- **Raspberry goes idle:**
 - after 1-minute raspberry turn off the screen and goes idle which makes it difficult to connect to it during debug
 - **solution:** is to disable the screen blackout, and idle state
- **Time sync:**
 - Raspberry isn't perfect in keeping the time accurate if Wi-Fi is used as most of the time, raspberry fail to get the ntp server response
 - **solution:** an exception for ntp server is added to raspberry ip table at boot time to mark this as high priority
- **Reading indicators from smart band:**
 - smart band providers don't introduce a way to read data from it programmatically instead you can read it from band app
 - **solution:** google fit is used, which we discovered later it's a better choice as it doesn't depend on manufactures
 - now we can read data from google fit API, which can be connected to almost any type of smart bands, watches and even mobile sensors
- **GPIO pins of the raspberry are not enough**
 - the number of GPIO pins of the raspberry don't enough raspberry has Raspberry Pi has 26 pin but we need around 30 pins.
 - we found difficult to find a raspberry GPIO breakout board so it will be hard and not safe to connect hardware direct.
 - **solution:** we used an Arduino board that can be controlled by raspberry pi with serial commands through USB cable.
- **tray initialization**
 - at first we used a limit switch to mark the initial position but the limit switch was Impeding the tray motion.
 - **solution:** we used an infrared object detection sensor and reflecting object on the tray to avoid friction.

- **acrylic sheet bending**

- while assembling the box we found the acrylic sheets bend due to weight of motor.
- **solution:** we used a supporting long metallic screws at bending points.

- **current sensor**

- we intended to use current sensor as feedback from air pump to know whether the pill is picked or not.
- unfortunately, while testing we found that pump current variation is too small to be used as trusted feedback.
- **solution:** instead of detecting whether the pill is picked or not, we depend on detecting whether the pill is dropped or not.
- we used an infrared object detection sensor to detect whether the pill is dropped or not, if not; the picking process is repeated again.

- **Power handling**

- stepper motors dissipate power even if it's not moving to brake.
- **solution:** we changed the code to cut the power off when stopped.

2. known issues

- **Raspberry-Firebase Authentication:**
 - Raspberry has no authentication to access firebase servers
 - as google doesn't provide an API for authentication for python
- **Raspberry WIFI handling:**
 - raspberry can't connect automatically to saved Wi-Fi unless it was available at boot time
 - this is a known bug in Raspbian and we can't do anything about it except making sure that the access point is available at boot time
- **Pump suction power isn't enough**
 - Pump can't catch all pills
 - Temporary solution: make the distance smaller between pills and hose, use small pills

3. Future Work

- Full control of the house "IOT applications".
- Make more compact design.
- Add more warehouses for medications.
- IOS app.
- Add schedule reminder to android app regardless whether it's connected to the box or not.
- Enhance dispensing mechanism.

4. Design parts

Box

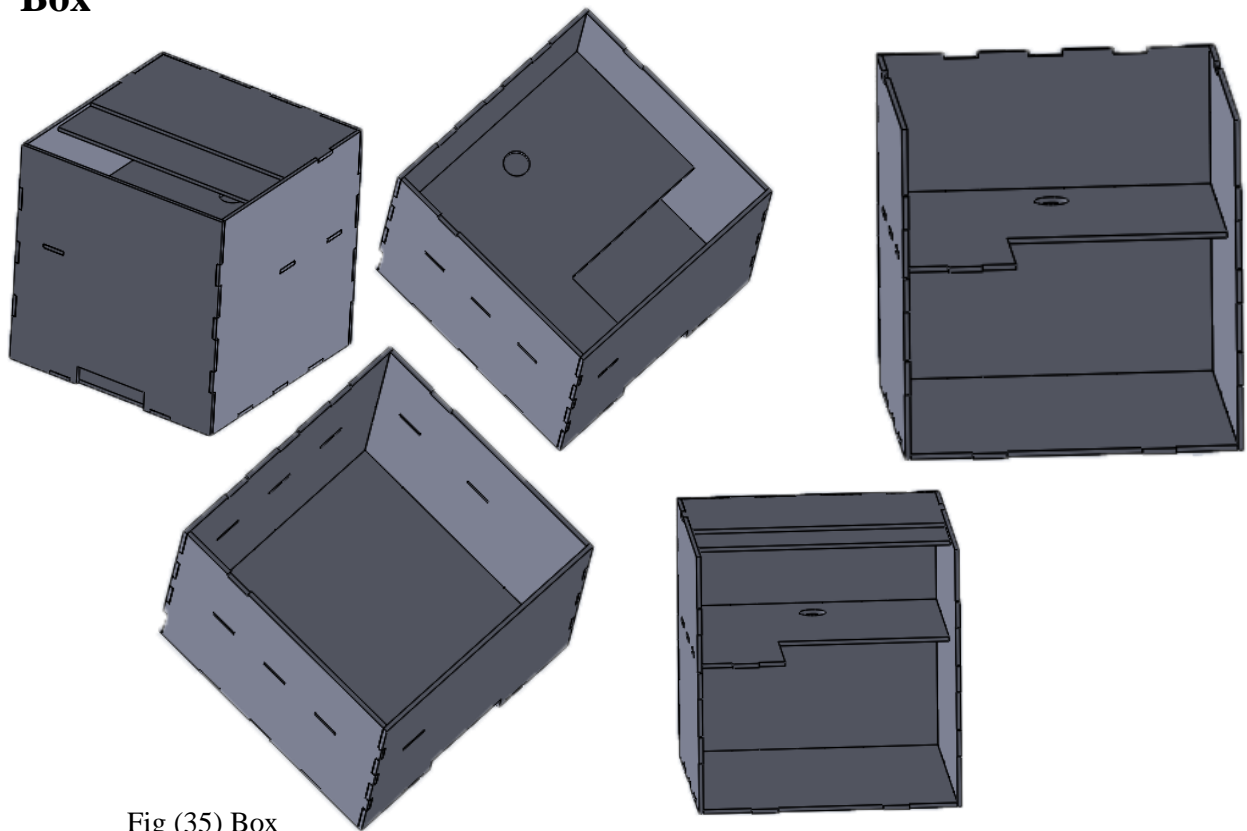


Fig (35) Box

Tray



Fig (37) Tray Mechanism

Drawer

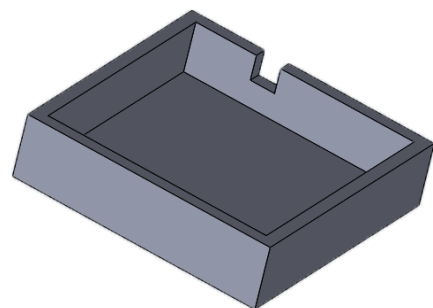


Fig (36) Drawer

5. Project Links

Android Client: <https://github.com/mahmoudShaheen/Medox>

RaspberryPi: Client: <https://github.com/mahmoudShaheen/PyMedox>

Cloud Function: <https://github.com/mahmoudShaheen/cloudMedox>

Arduino Code:

<https://github.com/mahmoudShaheen/PyMedox/tree/master/arduino>

6. License

Except as otherwise noted,
the content of this documentation is licensed under the Creative Commons
Attribution 4.0 International license (CC BY 4.0),
and code samples are licensed under the MIT License.
For more details, feel free to contact the owner.

The Creative Commons Attribution 4.0 International license (CC BY 4.0):



<https://creativecommons.org/licenses/by/4.0/>

10. Appendix

The MIT License (MIT)

Copyright © 2017 Mahmoud Shaheen

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7. Third party Notice:

All product names, logos, and brands are property of their respective owners. and are registered/unregistered trademarks of their respective owners and/or their affiliates.

All company, product and service names used in this documentation are for identification purposes only.

Use of these names, logos, and brands does not imply endorsement.

Used Google Documentations/Guides are licensed under the Creative Commons Attribution 3.0 License

<https://creativecommons.org/licenses/by/3.0/>

Project may contain code snippets and/or code samples from Google Inc. open Source snippets/projects which are Licensed under the Apache License: <http://www.apache.org/licenses/LICENSE-2.0>

Note: Detailed Notice files are included in each project's GitHub Repository.

10. Appendix

Medox includes some third-party python libraries to help it work.

Firestore-python:

<https://github.com/shariq/firebase-python>

sseclient "used by Firestore-python":

<https://github.com/btubbs/sseclient>

six "used by sseclient":

<https://github.com/benjaminp/six>

requests "used by Firestore python":

<https://github.com/kennethreitz/requests>

Emergency sound:

Uploaded: 05.17.11 | License: Public Domain | Recorded by TheCristi95

<http://soundbible.com/1819-Police.html>

Icons:

All resources available for download on GraphicLoads are royalty free for use in both personal and commercial projects.

Prohibitions:

You are prohibited to redistribute, resell, lease, license, sublicense or offer files downloaded from GraphicLoads to any third party.

Link: <http://graphicloads.com/license/>

contact: info@graphicloads.com

<http://graphicloads.com/product/medical-and-health-icons/>

8. References

- <https://stackoverflow.com/>
- <https://github.com>
- <https://firebase.google.com>
- <https://developers.google.com>
- <https://developer.android.com>
- <https://docs.python.org/2/library/>
- <https://nodejs.org/en/docs/>
- <https://www.raspberrypi.org/>
- <https://www.arduino.cc/>
- https://en.wikipedia.org/wiki/Stepper_motor
- <https://learn.adafruit.com/all-about-stepper-motors/what-is-a-stepper-motor>
- <https://www.circuitspecialists.com/stepper-motor>
- <https://www.circuitspecialists.com/blog/unipolar-stepper-motor-vs-bipolar-stepper-motors/>
- <https://store.fut-electronics.com/products/micro-servo-motor-1-3kg-cm>
- http://www.fut-electronics.com/wp-content/plugins/fe_downloads/Uploads/Introduction%20to%20Servo%20Motors%20&%20Arduino.pdf