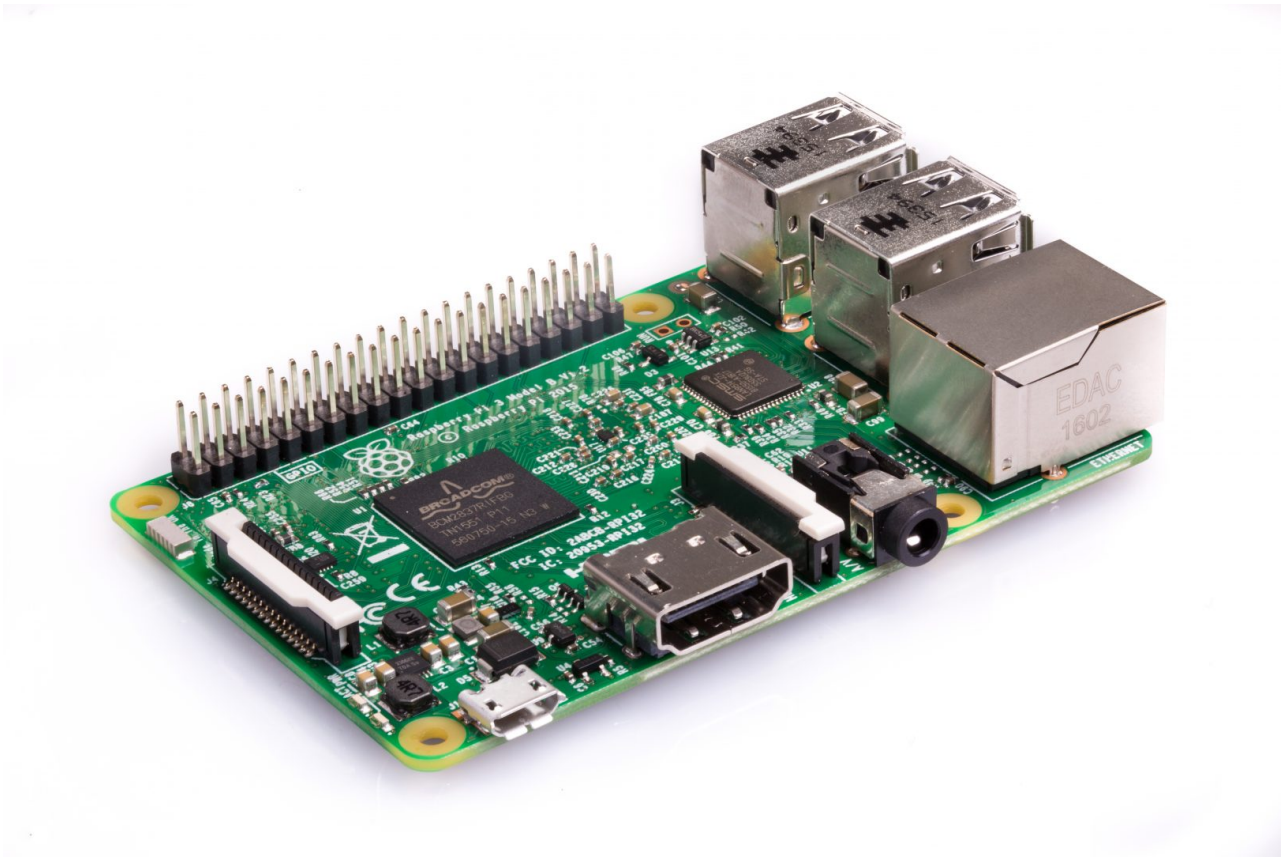
Home security system

Övervakningssystem med Raspberry Pi 3 model B

Växjö, 17-03-2018



Projektkurs i Datorteknik, VT 2018
2DT301
Handledare: Anders Haggren
Christoffer Roth

Sammanfattning

I dagens samhälle finns det en antal tekniska lösningar och tjänster på hemövervakning. Med varierande prisbilder och paket av tjänster så ställer sig detta arbete frågan, kan det vara ekonomiskt hållbart att utveckla ett eget övervakningssystem för tex hemmet? En stor massa äger idag en smartphone (eller surfplattor) och kraven på att kunna integrerar enheter via applikationer ökar. I detta arbete har ett hemsystem utvecklats där en sensor och kamera kopplas till en Raspberry Pi, Raspberry:n kommunicerar sedan med en egen utvecklad applikationen (för iOS) som är skräddarsydd för övervakningssystemet. Arbetet visar att det mycket väl kan vara ekonomiskt hållbart att utveckla sitt egna övervakningssystem, med det sagt så visar också slutsatsen att god teknisk kunskap krävs.

Innehållsförteckning

1. Introduktion

1.1 Bakgrund och problemformulering

I och med den utveckling som både hårdvara men även mjukvara har gjort dem senaste åren så har mer och mer enheter i form av olika elektronikprodukter blivit mer sammankopplade. Detta är en av faktorerna som driver utveckling i det som kallas Internet of Things (IoT) i rask takt framåt. Möjligheterna i IoT är t.ex hem som är uppkopplade och enheter i hemmet kommunicerar med varandra, detta kan göra att t.ex applikationer på smartphones blir ett gränssnitt för fjärrstyrning av ”smarta” hem.

Att få information och data från enheter i hemmet till sin smartphone i realtid kan ha många fördelar, några exempel på dessa kan handla om enheter som går sönder och behöver repareras, energiförbrukningen kan på ett effektivare sätt övervakas och regleras, det är även möjligt att installera ett övervakningssystem för övervakning av hemmet.

Övervakning av hemmet kan vara en stor trygghetsfaktor när hemmet står tomt. Men det kan även fungera som ett verktyg för att filma, ta bilder osv vid t.ex ett inbrott. Övervakningssystemet skulle kunna via smartphonen ge en notis att någon är i hemmet och dokumentera detta med film- och bildbevis. Skulle det finnas djur i hemmet så kan man med AI urskilja djur från personer och kunna minska risken för fellarm.

Idag finns det många företag som erbjuder kompletta lösningar av övervakningssystem/larm för hemmet. Med tillräcklig teknisk kunskap vad blir kostnaden och hur komplicerat är det att utveckla ett eget system? Är det hållbart rent kostnadsmässigt att köpa hårdvara och utveckla mjukvara till ett övervakningssystem själv?

1.2 Syfte och frågeställningar

Syftet med arbetet är att utveckla ett övervakningssystem som använder sig av en Raspberry Pi model B där en kamera och en avståndssensor kommer användas, raspberry:n kommer att kommunicera med en egen utvecklad iOS applikation. De huvudsakliga frågeställningarna för arbetet är:

- *Är det ekonomiskt hållbart att utveckla ett eget övervakningssystem för hemmet?*
- *Vilken teknisk kunskap krävs för att utveckla ett eget övervakningssystem?*

1.3 Avgränsningar

Projektet kommer att avgränsas till att endast utveckla en applikation för iOS och inte Android samt andra operativ system.

2. Metod

Projektet kommer kräva utveckling av programvara för både Raspberry och iOS. Raspberry:n programvaran kommer skrivas i python. Vad gäller iOS så är det Swift4 som är standardspråket och därav kommer applikation kommer utvecklas i detta språk.

Det som kommer behövas för att kunna spela upp en videoström i realtid från kameran är videoströmning via HTTP-protokollet. Det medför att en webbserver kommer behövas på Raspberry:n. Applikation för iOS kommer använda sig av sina bibliotek för att kunna ansluta till den givna URL (webbserver) och sedan spela upp denna videoström i en mediaspelare.

Ytterligare en server kommer att implementeras på raspberry:n, servern kommunicerar med MQTT-protokollet, Raspberry:n kommer då att fungera både som klient och server, applikation för iOS blir bara en klient. MQTT-protokollet är en standard för att skicka ”primitiva” meddelade mellan enheterna, detta kommer användas för att videoströmmen kan startas och stängas via applikationen samt att bilder kan tas.

Eftersom iOS inte kan spela upp h264 format via sin mediaspelare som kommer videoströmmen att strömmas som en MJPEG-ström. Då kameran på Raspberry:n kan spela och strömma i MJPEG-format blir detta första valet. Därefter kommer en mediaspelare för iOS att skapas som fångar upp bildströmmen och visa den via applikation.

3. Implementation

I detta projekt har mjukvara för både raspberry:n och iOS behövts utvecklas. På Raspberry:n har en webbserver hämtats som videoströmmar i realtid över HTTP, strömmen är i MJPEG-format. En applikation för iPhonen har utvecklas som dels skickar meddelade till Raspberry:n via ett IoT-protkoll (Internet of things) men också tar videoströmmen och spelar upp den direkt i applikationen.

3.1 Programvara Raspberry 3 model B+

Mjukvaran på Raspberry:n är utvecklad i Python (3.5). Python har ett antal stöd för moduler i sitt bibliotek som gör det praktiskt att arbeta med, då en sensor (modell HR-401) och en kamera (picamera model 2) är ansluten till Raspberry:n behöver programmet hantera data från dessa moduler. För att kunna steam video strömmen har en färdig webbserver importerats till programmet, webbserver är också skriven i python. Webbserver tar emot videon seglet inspelade i MJPG-format från kameran och skickat den via HTTP.

Mjukvaran använder sig av ett ytterligare protokoll för IoT, MQTT. Med detta protokoll kan Raspberry:n skicka meddelande om sin status till iPhonen och vice versa. Programmet kollar statusen på sensor med ca 2 sec intervaller, om avståndet skulle skilja med mer än 5 cm sedan tidigare mätningar så skickas ett meddelande från Raspberry:n till applikation i iPhonen. Ifall inte sensor ger utslag kan forfarande iPhonen skicka ett meddelande att Raspberry:n ska böra strömma video eller ta en bild. Så Raspberry:n är konstant ansluten till MQTT-server som också är på Raspberry:n, Raspberry:n är både en MQTT-client och en MQTT-server.

3.2 Programvara iOS

Programspråket för iOS är Swift 4 och all utveckling har gjorts i Apple:s egna text editor Xcode 9 (version 9.2). Designen av applikationen innehåller inga vidare avancerade funktioner, det som behövts att implementeras är ett antal knappar och ett textfält.

För att kunna styra raspberry:n via en iPhone så har en applikation utvecklats. Applikationen kommunicerar med raspberry:n via MQTT som är ett ”framework” som har importeras till projektet, iPhonen blir då en MQTT-client liknande Raspberry:n och de båda kan kommunicera med servern och skicka meddelande mellan varandra.

För att kunna spela upp videoströmmen från raspberry:n så har ett ytterligare framework importerats. Frameworket funktion är att ansluta till webbservers url och fånga upp videoströmmen, så länge webbserver har data att skicka så hålls anslutningen. Sedan används funktionen *ImageView* som är inbyggt i Swift 4, iOS programspråk, för att visa upp den inkoma videoströmmen.

4. Resultat och analys

I detta avsnitt presenteras resultat av arbetet.

4.1 Resultat

4.1.1 Resultat Raspberry Pi model B

För att kunna etablera kommunikation mellan raspberry:n och iOS-applikationen så har två open-source program som använder sig av MQTT-protokollet implementerats. Ett av programmen implementerar servern och det andra programmet implementerar klienten.

När programmet startas deklareras variabler, MQTT-klienten sättas upp och och ansluts samt anslutningen till webbservern för videoströmmen etableras. Figur 1 visar hur detta gör i programmet, MQTT-klienten ansluts och loppar så länge programmet körs, webbservern blir bara tilldelad IP och port och kommer att starts via kommunikation med iPhonen. Även kamerans resolution och framerate deklareras i detta läge. På rad 164 i figuren kalibreras avståndssensorn, sett görs via ett metodanrop och metoden kommer visas och förklaras senare i detta avsnitt.

```
134
135 ##########################
136 #                        #
137 # SETUP AND MAIN FUNCTION #
138 #                        #
139 ##########################
140
141
142 #Setup for the mqtt-client
143 mqttClientName = "RPI3B"
144 serverAddress = "192.168.0.104"
145 mqttClient = mqtt.Client(mqttClientName)
146
147 mqttClient.on_connect = connectionStatus
148 mqttClient.on_message = messageDecoder
149
150 mqttClient.connect(serverAddress)
151 print("Client connected to MQTT-server")
152 mqttClient.loop_start()
153
154 #Setup for video stream server
155 stream.output = stream.StreamingOutput()
156 address = ('', 8090)
157 server = stream.StreamingServer(address, stream.StreamingHandler)
158 server_streaming = False
159
160 #Camera setup
161 camera = picamera.PiCamera(resolution='320x240', framerate=24)
162
163
164 calibDistance = calibratedDistance()
```

Figur 1. Setup för MQTT-client, videoströmningen och sensorn

MQTT-protokollet låter klienter ansluta till en MQTT-server där klienter kan prenumerera på olika ”event”. När en klient har en händelse att rapportera så skickas ett meddelade till ett event och alla klienter som prenumererar på det eventet får meddelandet. Figur 2 visar hur och vilka event som Raspberry:n prenumerera på.

```
7  from time import sleep, time
8  from datetime import datetime
9  import picamera
10 import RPi.GPIO as GPIO
11 import paho.mqtt.client as mqtt
12 import cameraStream as stream
13 import threading
14
15
16
17 #Function that subscrubes the mqtt client to topics
18 def connectionStatus(client, userdata, flags, rc):
19
20     mqttClient.subscribe("rpi/cam")
21     mqttClient.subscribe("rpi/pic")
22     mqttClient.subscribe("rpi/dist")
23     mqttClient.subscribe("rpi/cls")
24
```

Figur 2. De event som MQTT-klienten prenumererar på

När iPhonen t.ex vill starta kameran så skickar den en ett meddelande via eventet *”rpi/cam”,* meddelandet blir avkodat till en sträng. Beroende på vad meddelandet innehåller så kommer antingen Raspberry:n starta videoströmmen, ta en bild eller i det fall videoströmmen är igång så kan den stängas. Nedan visar figur 3 vilken metod som blir kallad när ett meddelade annonseras på något av de prenumererade eveten. Metoden *messageDecoder* är en funktion som är en del av MQTT-klienten. Metoden blir kallad automatiskt när ett meddelande inkommer, meddelandet blir avkodat och strängen kan läsas.

```
25
26 #Function that get a msg from iphone and does something
27 def messageDecoder(client,userdata, msg):
28
29     message = msg.payload.decode(encoding='UTF-8')
30
31     if message == "StrtStream":
32         print(message)
33         cameraStream()
34
35     elif message == "TakePic":
36         print(message)
37         timestamp = datetime.fromtimestamp(time()).strftime('%-Y%-m%-s-%-H-%-M%-S')
38         camera.wait_recording(1)
39         camera.capture('/home/pi/Python/HomeSecurityPic/img{}.jpeg'.format(timestamp), use_video_port=True, resize=(1024, 768))
40         print("Picture is taken")
41
42     elif message == "CloseStream":
43         camera.stop_recording()
44         server_thread = threading.Thread(target=server.shutdown)
45         server_thread.deamon = True
46         server_thread.start()
47         server_streaming = False
48         print(message)
49
50     else:
51         print("Not a valid message")
52
```

Figur 3. Ett meddelande blir annonserat på ett av eventet. Meddelandet blir avkodat och läses.

Avståndssensor blir kalibrerad i uppstarten av programmet och mäts senare i en loop med ett par sekunders mellanrum. Skulle avståndet förändras med mer är fem cm så antas detta att sensors har upptäckt att t.ex en dörr har öppnats. Programmet skickar ett MQTT-meddelande via eventet *”rpi/dist”* till iPhonen i detta fall, där meddelas användaren att något har hänt.

Figur 4 visar hur sensorn mäter avståndet. Sensorn är mäter avståndet genom att skicka ut en puls (ultraljud) via TRIG, när pulsen skickas startas en timer som mäter tiden till ECHO uppfattar att pulsen har ”studsat" tillbaka. Tiden omformuleras till ett avstånd (i centimeter) genom multiplikation med en konstant, konstanten kan skilja lite beroende på omgivningen och känsligheten som önskas av sensorn. Avståndet returners från metoden, när en ny mättning ska göra så anropas metoden igen.

```
79  #Function that activates the sensor and takes its distance
80  def distSensor():
81
82      GPIO.setmode(GPIO.BCM)
83      GPIO.setwarnings(False)
84      ECHO = 25
85      TRIG = 24
86      GPIO.setup(TRIG, GPIO.OUT)
87      GPIO.setup(ECHO, GPIO.IN)
88      GPIO.output(TRIG, False)
89      sleep(2)
90
91      GPIO.output(TRIG, True)
92      sleep(0.00001)
93      GPIO.output(TRIG, False)
94
95      while GPIO.input(ECHO)==0:
96          pulse_start = time()
97
98      while GPIO.input(ECHO)==1:
99          pulse_end = time()
100
101     pulse_duration = pulse_end - pulse_start
102     distance = pulse_duration * 17150
103     distance = round(distance, 2)
104     GPIO.cleanup()
105
106     return distance
```

Figur 4. Sensorn skickar ut en puls och väntar på att den respons. Responsen mätts i tid och ett avstånds kan räknas ut.

Om en förändring av avståndet upptäckts så ska raspberry:n notifiera iPhonen om detta. Figur 5 visar hur raspberry:n gör detta, metoden *sendDistChangeMsg()* blir anropad och genom att använda MQTT-klient metoden *publish()* så skickas ett meddelade samt på vilket event som meddelandet ska skickas genom. Programmet läge i standby i 10 sekunder innan ett nytt avstånd kalibreras igen.

```
55  #Function that sends a message that the distance have changed to iphone
56  def sendDistChangeMsg():
57
58      mqttClient.publish("rpi/dist", "DistChanged")
59      print("Mail is sent, calibrate new distance")
60      sleep(10)
61      calibDistance = calibratedDistance()
62      print("Calibrated distance ", calibDistance)
63
```

Figur 5. När sensorn märker att avståndet ändrats skickas ett meddelande, ”DistChanged”, till applikationen via eventet ”rpi/dist”.

Meddelandet blir skickat till iPhonen och användaren bestämmer sig för att undersöka vad som har hänt, iPhone har två *publish()*-metoder likt Raspberry:n, hur detta ser ut i applikationens kod visas senare i detta avsnitt, iPhonen kan annonsera att Raspberry:n ska antingen ta en bild eller börja strömma en video. Förklarat och visat i tidigare figur (Figur 6) så tar Raspberry:n emot meddelandet. Ifall en bild ska bli tagen så görs det direkt i *messageDecode()*-metoden, bilden får ett ”timestamp” på när det blev tagen och sparas senare på Raspberry:ns minneskort. Om Raspberry:n ska börja strömma video istället så kalls metoden *cameraStream()*, metoden visas i figur X nedan. Metoden tar variablerna för servern som deklarerades vid uppstarten av programmet. När kameran början att spela in så ”sparas” videoströmmen ner i objektet *stream.output,* objektet skickas in till webbserver-klassen som bryter ner video till bytes och strömmar den över HTTP.

```
65  #Function that starts the streamserver and streams the video
66  def cameraStream():
67
68          camera.start_recording(stream.output, format='mjpeg')
69          print("+++Stream has started+++")
70          #try:
71          server_thread = threading.Thread(target=server.serve_forever)
72          server_thread.deamon = True
73          print("Server is starting")
74          server_thread.start()
75          server_streaming = True
76          print("Server thread running ", server_thread.name)
77
```

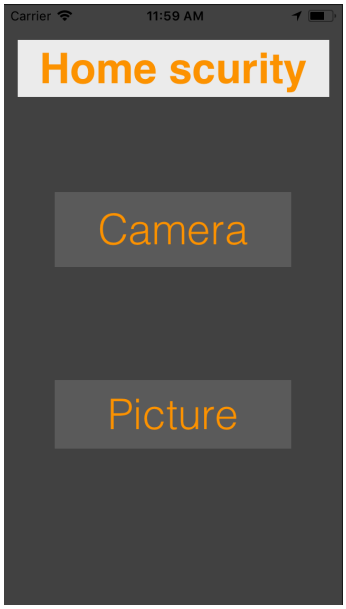
Figur 6. När meddelandet kommer om att starta videoströmmen gör ett metodanrop på cameraStream().

4.1.2 Resultat applikation för iPhone

För att en användare ska kunna kommunicera med Raspberry:n så har en applikation till iPhone utvecklats. Applikationen är utvecklad i programspråket Swift 4 och fungerar som en MQTT-klient. Detta görs med ett open source program som använder sig av MQTT-protokollet. För att kunna spela upp videoströmmen används ytterligare ett open source program som sköter anslutningen och hämtningen av MJPG-videoströmmen.

Figur 7 visar designen på applikationen, applikationen har två ”knappar”, en knapp för att skicka ett meddelande till raspberry:n att börja strömma video samt spelar upp videoströmmen. Den andra knappen

skickar till Raspberry:n att ta bild som sparas på raspberry:n. Hur de är implementerade kommer visas senare i avsnittet.



Figur 7. ”Index”-sidan för applikationen

Applikationen har precis som Raspberry:n ett antal variabler som deklareras vid uppstarten, Swift har också en setup metod som kör när appen startar. Figurerna 10 och 11 visar hur detta görs, i figur 10 görs en setup för MJPG-spelaren, i figur 11 görs dels en setup för att skicka en notification när raspberry:n skickar ett meddelande om att sensorn aktiverats, en setup görs också för MQTT-klienten där den saluter till MQTT-servern på raspberry:n. Koden på raderna 59 - 61 är en setup för fönster där videoströmmen kommer spelas upp.

```
9  import UIKit
10 import MjpegPlayer
11 import CocoaMQTT
12 import UserNotifications
13
14 class ViewController: UIViewController {
15
16
17     @IBOutlet weak var Background: UIImageView!
18     @IBOutlet weak var HomeLabel: UILabel!
19     @IBOutlet weak var Camera: UIButton!
20     @IBOutlet weak var Picture: UIButton!
21     @IBOutlet weak var StreamClose: UIButton!
22
23     /*
24      * Variables for the MJPG-stream
25      */
26     private var streamingController: MjpegStreamingController!
27     private var imageView: UIImageView!
28     private let raspURL = URL(string: "http://192.168.0.104:8090/stream.mjpg")
29
30     /*
31      * Variables for the MQTT-server
32      */
33     var mqtt:CocoaMQTT?
34
35     /*
36      * Setup function
37      */
```

Figur 10. Setup för MJPEG-strömmen samt en global variabel för MQTT-klienten

```
34
35      /*
36       * Setup function
37       */
38      override func viewDidLoad() {
39          super.viewDidLoad()
40          // Do any additional setup after loading the view, typically from a nib.
41
42          UNUserNotificationCenter.current().requestAuthorization(options:
                ([.alert, .sound, .badge]), completionHandler: {didAllow, error in
43              if didAllow {
44                  print("Granted")
45              }
46              else {
47                  print(error?.localizedDescription as Any)
48              }
49          })
50
51          /*
52           * Variables for the MQTT-server
53           */
54          let clientID = "iOSDev-" + String(ProcessInfo().processIdentifier)
55          mqtt = CocoaMQTT(clientID: clientID, host: "194.47.179.134", port: 1883)
56          mqtt!.delegate = self
57          mqtt!.connect()
58
59          imageView = UIImageView(frame: CGRect.init(x: 0, y: 100, width: 320, height:
                240))
60          self.view.addSubview(imageView)
61          imageView.isHidden = true
62
63      }
64
```

Figur 11. Setup för notifikationen som ska annonseras vid sensor utslag och MQTT-klienten ansluts till MQTT-servern.

MQTT-klinten fungerar mer eller mindre på samma sätt som på Raspberry:n. Klienten väntar på att ett meddelande annonseras på något event. Det event som applikationen lyssnar på är meddelande från Raspberry:n när sensorn har fått utslag. När meddelades angående sensor har mottagits så görs en anrop på metoden som skickar en notis till användaren. Övrig kommunikation sker genom att applikationen skickar meddelande till raspberry med en liknande metod som används på Raspberry:n, *publish()*. Figur 12 visar hur detta ser ut i kod.

```
168
169     func mqtt(_ mqtt: CocoaMQTT, didConnectAck ack: CocoaMQTTConnAck) {
170         if ack == .accept {
171             mqtt.subscribe("rpi/dist", qos: CocoaMQTTQOS.qos1)
172             print("Connected and subscribed to rpi/dist")
173         }
174     }
175
185     func mqtt(_ mqtt: CocoaMQTT, didReceiveMessage message: CocoaMQTTMessage, id: UInt16)
            {
186
187         let string = message.string!
188         print(string, "Recive")
189
190         if string == "DistChanged" {
191             print("Setup notification")
192             sendNotification()
193         }
194     }
```

Figur 12. Applikationen prenumerera på eventet ”rpi/dist”. När ett meddelande ankommer avkodas det.

Metodanropet på *sendNotification()* gör när ett meddelandet *”DistChanged”* kommit in från raspberry:n, metoden visas i figur 13. Metod skickar ut en notis på telefonen att sensorn har aktiverats. Detta var inte problemfritt och denna funktion fungerar ej som tänkt. Notisen ska skickas till även om applikationen ej är aktiv, denna funktion har ej kunnat implementerats. Applikationen skickar bara notisen ifall applikationen är öppen. En lösning på detta problem har inte hittats så funktionen för notisen lämnas till och används i dagsläget endast ifall applikationen är öppen.

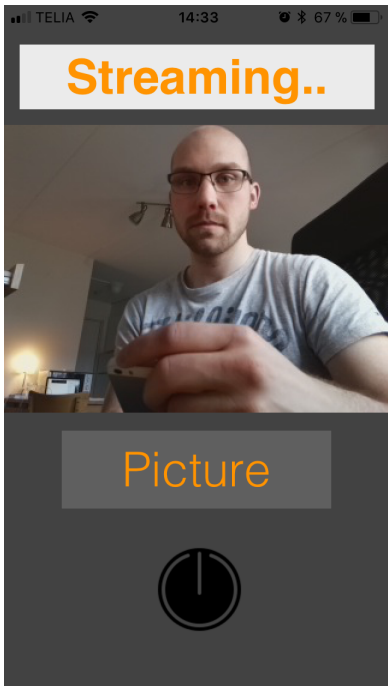
```
137
138     func sendNotification() {
139
140         let content = UNMutableNotificationContent()
141         content.title = "Sensor has been activated"
142         content.subtitle = "The calibrated distance have been changed"
143         content.body = "Sensor has been activated"
144
145         let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 2, repeats: false)
146         let request = UNNotificationRequest(identifier: "DistChange", content: content,
                trigger: trigger)
147
148         UNUserNotificationCenter.current().add(request, withCompletionHandler: nil)
149
150     }
151
```

Figur 13. Metod som skickar ut en notifikation till användaren.

De två huvudsakliga funktionerna i applikationen är att kunna spela upp en videoström och ta bilder. Vad gäller videoströmmen så skickar applikationen ett meddelande till raspberry:n att börja spela in och strömma video. Figur 14 visar metoden som blir anropad när *”Camera”* är tryckt, meddelandet skickas och MJPG-spelaren fångar upp videoströmmen. Videoströmmen blir uppspelad i det fönster som blir deklarerat i början av programmet (figur 15), MJPG-formatet skickar jpeg bilder i en ström. *ImageViewer* fångar up bilderna och spelar upp dom, figur X visar en print screen på hur detta ser ut (en videoklipp är bifogat med rapporten).

```
75
76      /*
77       * Function that reacts to camera button when pressed.
78       * Send to the raspberry to start stream and the play the stream.
79       */
        @IBAction func Camera(_ sender: UIButton) {
81
82          HomeLabel.text = "Streaming.."
83          mqtt!.publish("rpi/cam", withString: "StrtStream")
84
85          sleep(UInt32(1.0))
86
87          playMJPEGVideoStream()
88
89      }
90
91      /*
92       *  Function that starts to play the the stream from the
93       *  the raspberry camera.
94       */
95      func playMJPEGVideoStream() {
96
97          StreamClose.isHidden = false
98          imageView.isHidden = false
99
100         self.view.addSubview(imageView)
101         streamingController = MjpegStreamingController(imageView: imageView, contentURL:
                raspURL!)
102         streamingController.play()
103
104     }
105
```

Figur 14. Meddelande skickas till raspberry:n att starta videoströmmen och strömmen hämtas av MJPEG-spelaren och visas i ImageViewer:n.



Figur 15. Videoströmmen visas i ImageViewer:n.

Funktionen för att stänga videoströmmen fungerar på samma sätt som *”Camera”* och *”Picture”*. Ett meddelande skickas till Raspberry:n som då stänger ner inspelningen och webbservern. Här kan även sägas att både avstängningsknappen, resterande knapparna och videofönstret är byggt som ”lager”, dessa lager kan styras genom att döljas eller synas. Detta gör med funktionen *isHidden*, funktionen används i de flesta metoderna i applikationen. I figur 16 visas funktionen som anropas när avstängningsknappen blir tryckt.

```
105
106     /*
107      *  Function that closes the stream when the the
108      *  close button is pressed. Also sends message to
109      *  the raspberry that the stream should be shutdown.
110      */
        @IBAction func StreamClose(_ sender: UIButton) {
112
113         mqtt!.publish("rpi/cls", withString: "CloseStream")
114         print("Sent close stream to raspberry")
115
116         sleep(UInt32(1))
117
118         streamingController.stop()
119         StreamClose.isHidden = true
120         imageView.isHidden = true
121         Picture.isHidden = false
122
123         HomeLabel.text = "Home security"
124     }
125
```

Figur 16. Metoden skickar att videoströmmen ska stängas och anslutningen bryts.

Den sista funktionen (figur 17) som finns i applikationen är funktionen att meddela raspberry:n att fånga bilder. Bilder kan kan bli tagna både när videoströmmen är avstäng och igång. *”Picture”*- knappen är därför alltid aktiv så användaren har möjlighet att meddela ta bilder ska bli tagna. Applikationen skickar altid *”rpi/pic”* för att meddela Raspberry:n att ta bilder, Raspberry:n löser bildtagning beroende på ifall videoströmmen är ingång eller avstängd.

```
65
66      /*
67       * Function that reacts to picture button when pressed.
68       * Send to the raspberry to take a picture and send it to the iphone.
69       */
        @IBAction func Picture(_ sender: UIButton) {
71
72          HomeLabel.text = "Taking Pictures"
73          takePicture()
74      }
75
76      /*
77       * Function that gest a picture from the raspberry camera.
78       */
79      func takePicture() {
80
81          mqtt!.publish("rpi/pic", withString: "TakePic")
82          HomeLabel.text = "Picture capturead"
83          sleep(UInt32(0.5))
84          HomeLabel.text = "Home security"
85
86      }
87
```

Figur 17. Skickar meddelande till raspberry:n att en bild ska fångas.

4.2 Analys

Det är tydligt från resultatet att en stor del projektet och övervakningssystemet bygger på kommunikationen mellan applikationen och raspberry:n. MQTT-protokollet kommer väl till hands för detta projekt och det har visat sig vara väldigt användbart. Användningen av open source programvara kan vara risk då support och buggar kan utebli. I detta fallet så användes ändå open source program då MQTT är ett populärt protokoll och programmen som används har bra stöd.

5. Diskussion och slutsatser

I det avslutande avsnitten så reflekterar jag över det get gjorda arbete och dess slutresultatet. Utmaningarna i arbetet var inte enbart att fundera ut ett lämpligt projekt utan planering, uppföljning och det faktum att jag självmant valde att göra två mjukvaror i skilda programspråk (python och Swift 4) som jag aldrig jobbat med innan.

5.1 Diskussion

Målet med projektet var att utveckla en övervakningssystem för hemmet. I mitt fall utgick det från att avhända sig av en sensor som tex riktades mot en dörr, när dörren öppnas aktiveras sensorn. En kamera används för att kunna ta bilder eller video övervaka. För att ge övervakningssystemet användarvänlighet så krävdes någon form av kommunikation från sensorn och kamera till de boende, till detta användes en Raspberry Pi och en egen utvecklad applikation för iPhone.

Mjukvaran för raspberry är som sagt utvecklad i python, detta dels för att programmera i ett nytt språk. Det visade sig även att många open source program var skrivna i python, även bibliotek för kameran och I/O-pinsen fanns i python. Detta gjorde att jag hade mycket material att använda som referens och inkörsporten till språket blev därav relativt kort.

Programvarorna implementerar MQTT-protokollet visade sig också vara väldigt användbart. Även dessa programvaror var skrivna i python vilket gjorde användningen av programvaran betydligt lättare. Det hade varit möjligt att utveckla ett eget protokoll för att hantera kommunikationen mellan enheterna. Men utmaningen med att dels jobba med två nya programspråk sen även de svårigheter som uppstod med etablera och spela upp videoströmmen så togs beslutet att använda färdig programvara. Som sagt, programvarorna implementerar MQTT-protokollet vilket även gör det möjligt att ansluta fler enheter utan att göra några förändringar på de befintliga enheterna.

Det som visade sig vara den största utmaningen och den funktion som tog längst tid att implementera var att spela upp en videoström i realtid. De svårigheter som detta presenterade var inte bara hur videoströmmen skickas till iPhonen från Raspberry:n utan även hut videoformatet skulle se ut. För att kunna spela upp video i real tid behöver iPhonen få strömmen via HLS(HTTP Live Streaming), enklare beskrivning är att iPhonen behöver få videoströmmen över HTTP-protokollet. För att strömma över HTTP så har en extra python script hämtas (picamera docs), scriptet tar en output som i detta fall är en videoström i MJPEG-format och strömmar det via HTTP via sockets. Detta gör att en TCP-connection startas med iPhonen när videoströmmen ska spelas i applikationen. MJPEG-formatet är jpeg-bilder som är tagna och skickade (framerate [bilder/sec] = 24). Strömmen spela upp i applikationen genom att ”en” jpeg-bild hämtas ur strömmen, visas och sen plockas nästa bild ut ur strömmen osv. Fördelen med denna lösning är att ingen formatering behövs göras utan det som behövs är att applikationen plockar upp bilderna och helt enkelt visar dem som jpeg:s. I applikationen sköts detta med ett framework (importerad program), framework:t sköter anslutningen till den webbservern som HTTP-strömmen kan hämtas ifrån. Framework:t tar emot bilddatan, konverterar det till en image sen returneras imagen:n till det fönster som jag har implementerar med *ImageView*. På frågan om detta hade gått att göra en egen lösning för detta så är svaret givetvis ja, dock blev det uppenbart ganska snabbt att det var inte ett optimalt första steg i applikationsprogrammering.

5.2 Slutresultat

Resultatet av projektet visar att det är fullt möjligt att utveckla ett eget övervakningssystem för hemmet. Raspberry:n visar sig vara mycket användbar och det finns mycket stöd och resurser för både mjukvara och hårdvaran. Men jag skulle ändå vilja påstå att god teknisk kunskap behövs, mycket av den mjukvara som finns tillgänglig att implementera på Raspberry:n är open source. Open source kräver att man har god kunskap om programvara vad gäller säkerhet och uppdateringar osv. Ifall man väljer att utveckla all

mjukvara från grunden så behövs god, t.o.m mycket god kunskap om programmering och goda kunskap inom datorområdet.

Kostnaden för projekt, iPhonen är borträknad, landade på totalt 850 kr. I kostnaden räknades, Raspberry Pi model 3, PiCamera module V2 och minneskort på 16 GB. På frågan om detta är *"..ekonomiskt hållbart?"* så skulle jag vilja säga ja. Jag anser att detta är ett rimligt pris och kostnaden för fler sensorer eller kameror behöver inte skjuta iväg kostanden för ett större system till ohållbara summor.

Här följer övergripande slutsatser som resultat av projektet:

- Det är fullt möjligt att utveckla ett eget övervakningssystem för hemmet där man kan styra enheter via en smartphone.
- Det krävs god teknisk kunskap för att genomföra denna typ av projekt.
- Det är ekonomiskt hållbart att utveckla ett eget övervakningssystem.
- Det är möjligt att ansluta nya enheter till systemet och styra dessa med applikationen.

Referenser

1. http://picamera.readthedocs.io/en/latest/recipes2.html#web-streaming. Hämtad 16/3-2018