Medibot

Written by Jake Davis, Luke Bonner & Euan Gilfillan

Funded by Inside Pensions

The Priory School, Hitchin

Table of Contents

Project Overview and Aims 3

Design 4

Technical Information 5

Robot specification 7

Evaluation 10

Programming 11

- Code for the Control Panel 11
  - index.html 11
  - app.css 12
  - app.js 13
- Code for the Robot 16
  - app.py 16
  - camera.py 19

Project Overview and Aims

Over 3 and a half million people over the age of 65 live alone, and of these according to Age UK, 50,000 use their pendant alarm system that allows them to live independently (see Figure 1 Traditional Pendant Alarm System). Although we do not think that the traditional alarm system is sufficient for these people as when the alarm is activated the help centre cannot fully establish what assistance the person requires and if any further medical attention is needed.

To respond to this problem, we have created a robot to be kept in houses with elderly and disabled inhabitants to be used to give people independence in their homes.

Medibot is designed to provide contact between vulnerable people and a help service, should they fall into difficulty and require assistance. Medibot works as a camera between the call centre and the client, should the Medibot ever be activated the employee at the call centre would be able to control the device, be able to assess the situation and execute the correct protocol.

Our idea originated when we saw a weakness in the panic button-style system that is currently in place in many care homes and the homes of individuals around the country. When the button is pressed, a person who can deal with requests like these is notified and must make a call on further action with no information at all from the person's current circumstances, due to the button's design. Medibot would continue to use the button for activation as we find that it is a very efficient and easy way for activation.

We were motivated to do the project due to the recent NHS budget cuts. This meant that fewer people could be live in full time accommodation at care homes and would therefore have to live independently. Our project would be able to provide safety to these people and to help people if they are ever in danger.

Figure 1 Traditional Pendant Alarm System

Design

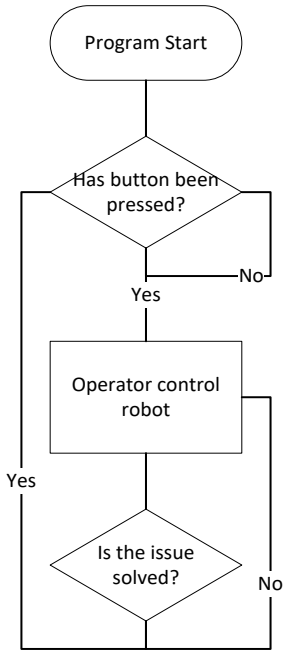
There are three key elements of our project which we aimed to complete whilst developing the robot: Firstly, the robot had to be robust. If it was deployed to people's homes, the robot must always work. It is central that the robot is reliable and has effective redundancy methods that meet the standards of the traditional button system.

Secondly, the button needs to be simple and accessible for the client to activate. Both the software and hardware has to be easy to use for the client and help service so that sufficient help is supplied quickly.

Our last key element is the safety of the product. Medibot must be safe to use in the house and not be a cause of harm. The device must not affect the way a person would normally live their life and therefore not be an inconvenience.

The process the robot undergoes is shown in Figure 2.

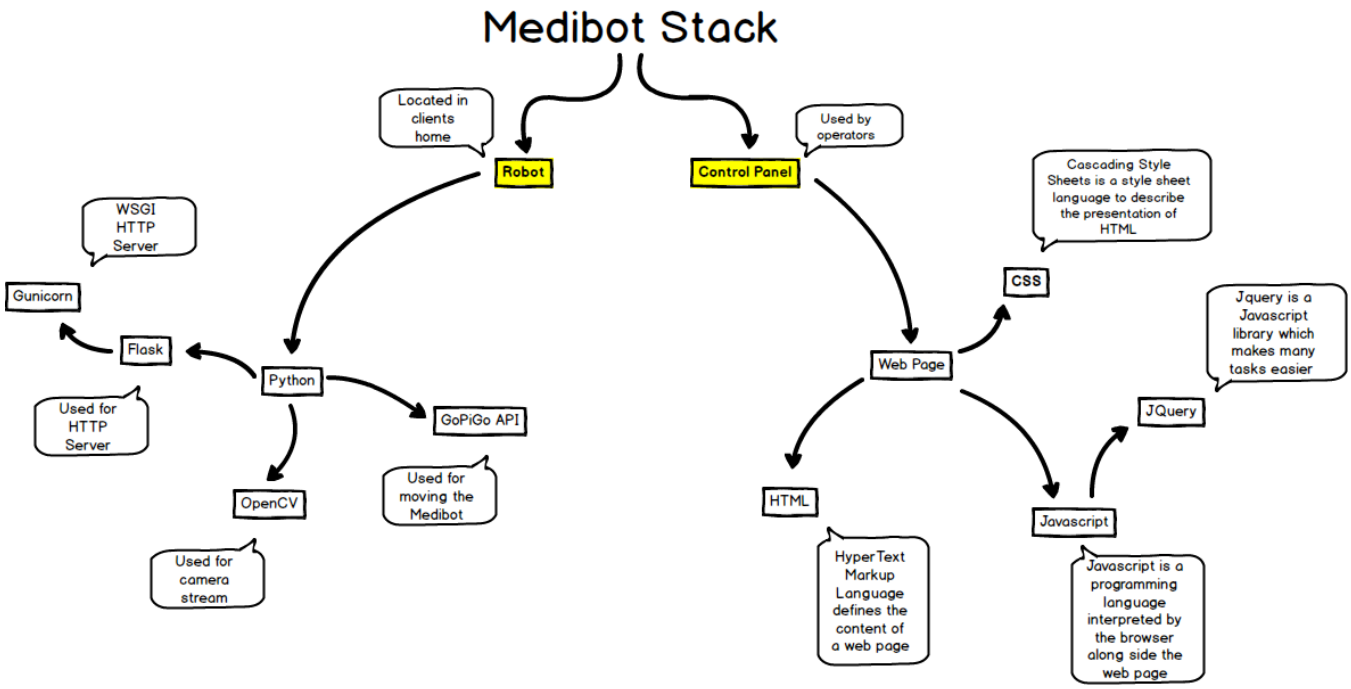
Figure 2 Initial Plan



Technical Information

Figure 3 shows the stack of our Medibot. Our stack is the pieces of software and the tools we used to achieve our aims.

Figure 3 Diagram showing Medibot stack



Executing commands on the robot is done by hosting a HTTP server on the Medibot. This works in a similar way to a website; when the page "/move?drcn=forward" is accessed the robot will begin to move forward. To create this HTTP server, we used Flask - a Python module that makes serving these types of requests simple. The 7 lines of code in the documentation for Flask below is a "Hello World!" example for the module (more examples of these requests can be found on page 11).

Figure 4 Screenshot from Flask documentation

Flask 0.12.2

A microframework based on Werkzeug, Jinja2 and good intentions

Flask is a microframework for Python based on Werkzeug, Jinja 2 and good intentions. And before you ask: It's BSD licensed!

Flask is Fun

Save in a hello.py:

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

The movement of the robot is controlled by using the GoPiGo API, this is a module developed by the manufacturer of the GoPiGo board which allows the handling of the servo (a device that can be set to rotate the object fixed to it at a certain degree) and motors. Which provides a simple method of handling the devices.

We also needed to provide a camera stream, we did this by using the OpenCV module. OpenCV helps us create a video camera object where the frames which the stream is composed of are sent though Flask. Initially we used MJPEG streamer for the camera stream but we found that it was to slow as there was a 5 second latency time (latency in this case is the time delay between what image is outputted from the camera and when it is shown on the control panel).

The control panel is a website composed of HTML, CSS and Javascript. HTML is a mark-up language that defines the content of the page, CSS was then used to stylise this content. The CSS framework called Foundation helped us to design the layout of the control panel. Javascript (abbreviated to JS), is a programming language interpreted by the browser, was used to add interactivity to the website.

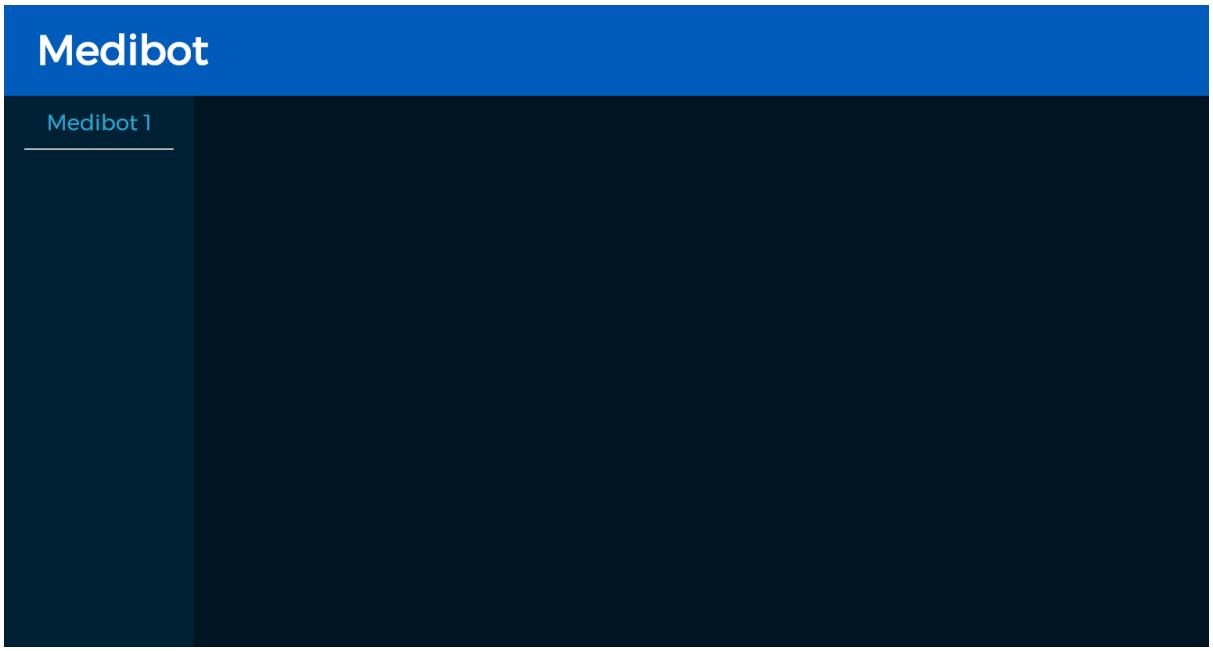


Figure 5 Screenshot of control panel

A JS library called JQuery has been used to detect keypresses to move the robot whilst on the control panel. This is a library of written Javascript which allows for easier development of Javascript based applications.

The control panel works by sending requests to the Medibot when any keys are down, e.g. if the up arrow is down "/move?drcn=forward" will be accessed which will make the robot move forward. This is also the same for the servo and any other direction.

As mentioned in the page 3, the Medibot hosts a stream of the camera. This is viewed on the top right of the control panel. This camera stream acts as a collection of JPEG images and can be shown by using an image tag in html, "<img>".

Robot specification

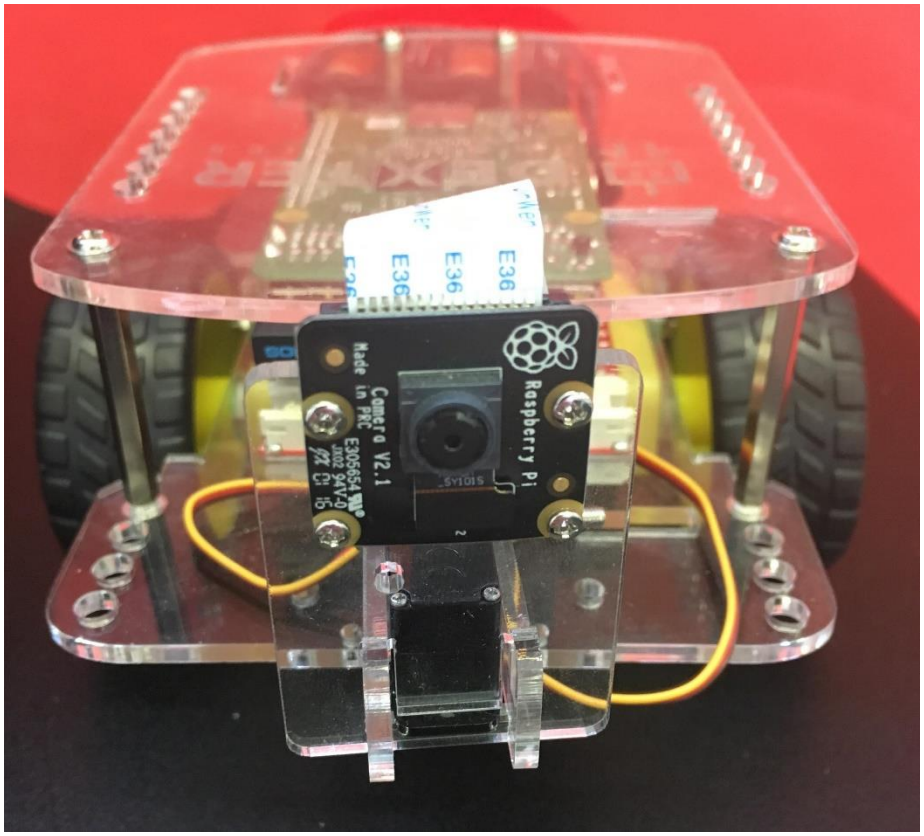
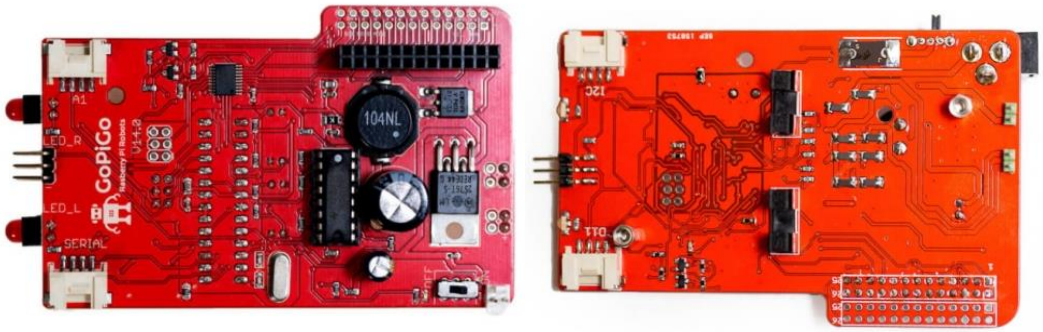


Figure 6 The robot we constructed composed of the parts described below

To make the Medibot, we used a variety of devices and instruments which all serve a purpose as a part within our robot. Images of these devices are shown below with a brief explanation as to what each part is and what it does.

- **GoPiGo circuit board and motor controller** - this controls the servo and motors though the Raspberry Pi that this board is fastened to.



- **Raspberry Pi 2 B** – A microcomputer which stores and executes the code, it also uses the GoPiGo circuit board to control the servo and motors. All peripherals below excluding the servo are connected to it.

- **Camera Servo** – This servo controls the rotation of the camera which is fastened onto the chassis. It is connected to the GoPiGo instead of the Raspberry Pi.

- **Raspberry Pi NOiR camera** – This camera can provide an image when in the dark, this is central in responding to problems at night time.

- **Wi-Fi dongle** – This USB device connects the robot wirelessly to the remote, control panel and allowed us to use VNC and SSH. VNC stands for virtual network computing and which uses remote desktop into our robot. Likewise, SSH allows for remote access to our robot in a terminal.


802.11n

Evaluation

We think that our robot achieved its aims of allowing visual communication between the emergency call centre and the vulnerable people but we would need to continue with developing it so that it could be deployed into people's homes and also replace the current alarm system.

We used Python as the programming language for our HTTP server as we already had experience in the syntax of the language. We also found it the most effective because it had an array of modules we could use to help us develop the software.

We think that our program could be improved with the use of sockets instead of the use of a HTTP server. This would increase the speed and reliability of communication between the control panel and Medibot and would therefore be helpful to implement.

During the development of our project we have changed the software stack of our project. We were previously using the .NET Framework for the control panel but concluded that a web based control panel would be more effective because of it being able to run/interpreted on multiple platforms.

As there is a scope to do more we would develop a docking station to charge the robot and provide audio capabilities. This would allow for increased communication between the vulnerable person and the emergency call centre as well as lowering maintenance because of automatic charging capabilities.

To improve on the safety of the robot we would like to develop an outer casing. This would protect the person as well as the robot. We would make it brightly coloured for any partially blind people so that there is increased visibility. www.visionaware.org states that “Solid, bright colours, such as red, orange, and yellow are usually more visible than pastels.” so we would therefore aim to create a red casing.

We think that our project has a real-world application as it has the ability to help thousands of vulnerable people live more independently.

Programming

Code for the Control Panel

index.html

This HTML file defines the structure of our control panel. It identifies every element to be shown on the web page. Most elements have classes and id's which are used to categorise and identify the different elements.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta http-equiv="x-ua-compatible" content="ie=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Medibot</title>
  <link rel="stylesheet" href="css/foundation.css">
  <link rel="stylesheet" href="css/app.css">
</head>
<body>

<div class="top-bar">
  <div class="top-bar-left">
    <div class="menu-text">Medibot </div>
  </div>
</div>

<div class="row align-top align-justify container">
  <div class="columns robot-list-column shrink">
    <ul class="menu robot-list-panel">
      <li>
        <a href="#">Medibot 1</a>
        <hr class="small-hr">
      </li>
    </ul>
  </div>
  <div class="columns control-panel" id="information">
```

```
  </div>
  <div class="columns camera-panel small-3">
     <img class="camera-stream" src="http://192.168.137.221:5000/video_feed"
onclick="streamFullscreenToggle()">
  </div>
</div>

<script src="js/vendor/jquery.js"></script>
<script src="js/vendor/what-input.js"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/mustache.js/2.2.1/mustache.js"></script>
<script src="js/vendor/foundation.min.js"></script>
<script src="js/app.js"></script>

</body>
</html>
```

app.css

This CSS file stylises the content defined in index.html.

```
@import url('https://fonts.googleapis.com/css?family=Montserrat');

.top-bar {
 background-image: linear-gradient( 135deg, #3AADFF 0%, #0396FF 100%);
}

* {
 color: #fff;
 font-family: 'Montserrat', sans-serif;
}

body {
 background-color: #014677;
}

.robot-list-column {
 background-color: #0168b2;
}
```

```
.camera-panel {
 background-color: #0168b2;
}

.columns {
 height: 100%;
}

.row {
 margin: 0;
 max-width: none;
}

.container {
 height: 100%;
 width: 100%;
}

.small-hr {
 margin: 0 auto;
}

.camera-stream {
 width: 100%;
 height: auto;
 display: block;
 margin: 10px auto;
}

.fullscreen {
 z-index: 9999;
 position: fixed;
}
```

app.js

This is the Javascript file which adds the content to the web page and controls the robot by sending HTTP requests to it.

```
$(document).foundation();
```

```
var domain = "http://192.168.137.221:5000/";
var cameraStream = $(".camera-stream");
var cameraStreamHeightAuto = $(cameraStream).height();
var cameraStreamOffset = $(cameraStream).offset();
var cameraOriginalWidth = $(cameraStream).width();
var servoPos = 90

/* RENDER INFO */
$.get('./templates/information.mst', function(template) {
  $.getJSON(domain + "get_info", function (data) {
      console.log(data);
      var rendered = Mustache.render(template, {emergencyContacts: data['contacts'],
                                        name: data['information']['name'],
                                        dob: data['information']['dob'],
                                        illness: data['information']['illness'],
                                        medication: data['information']['medication']});
      $('#information').html(rendered);
  });


});


/* CONTROLLING */
$.getJSON(domain + "servo?pos=90", function () {
  console.log("servo?pos=90")
})

var lastKey = ""

$(document).keydown(function(event) {
      // CONTACT SERVER
      if (event.key == lastKey) {
          console.log("MOVE " + event.key);
          if (event.key == "s") {
              action = "move?drcn=backward"
          }
          if (event.key == "w") {
              action = "move?drcn=forward"
          }
```

```
        if (event.key == "d") {
            action = "move?drcn=right"
        }
        if (event.key == "a") {
            action = "move?drcn=left"
        }
        if (event.key == "ArrowLeft") {
            if (servoPos < 177) {
                servoPos += 3
            }
            action = "servo?pos=" + servoPos
        }
        if (event.key == "ArrowRight") {
            if (servoPos > 3) {
                servoPos -= 3
            }
            action = "servo?pos=" + servoPos
        }
        if (event.key == "ArrowUp") {
            servoPos = 90
            action = "servo?pos=90"
        }
    }

    lastKey = event.key
    $.getJSON(domain + action, function () {
        console.log(action)
    })
});

$(document).keyup(function() {
  console.log("STOP");
  $.getJSON(domain + "stop", function () {
      console.log("STOPPED")
  });
});

/* STREAM */

function streamFullscreenToggle() {
```

```
  if(!$(cameraStream).is(':animated')) {
    if (!$(cameraStream).hasClass('fullscreen')) {
      $(cameraStream).addClass('fullscreen');
      $(cameraStream).animate({
        width: "100%",
        height: "100%",
        top: 0,
        left: 0
      }, 1000);
    } else {

      $(cameraStream).animate({
        width: cameraOriginalWidth,
        height: cameraStreamHeightAuto,
        top: cameraStreamOffset.top,
        left: cameraStreamOffset.left
      }, 1000, function () {
        $(cameraStream).removeClass('fullscreen');
        $(cameraStream).css('top', 'auto').css('left', 'auto');
      });
    }
  }
}
```

Code for the Robot

app.py

This is the python file which hosts the HTTP server for controlling the robot. It also streams the content of the camera provided by camera.py below.

```
from flask import Flask, send_file, json, request, Response
from flask_cors import CORS, cross_origin
from gopigo import *
from subprocess import Popen, call
import os
import csv
from threading import Thread
from time import sleep
```

```
from camera import VideoCamera

app = Flask(__name__, static_folder='/')
CORS(app)

stop()
enable_servo()
led_off(1)


def siren():
    while True:
        led_on(1)
        led_off(2)
        time.sleep(0.25)
        led_on(2)
        led_off(1)


@app.route('/get_info')
def get_info():
    emergency_contacts = csv.DictReader(open('data/emergency_contact.csv.txt'))
    medical_information = csv.DictReader(open('data/medical_information.csv.txt'))

    emergency_contacts_list = []
    for item in emergency_contacts:
        emergency_contacts_list.append(item)

    medical_information_list = []
    for item in medical_information:
        medical_information_list.append(item)

    return json.dumps({"information": medical_information_list[0], "contacts":
emergency_contacts_list})

@app.route('/move')
def move():
    direction = request.args.get('drcn')
    if direction == "forward":
        motor_fwd()
```

```
    elif direction == "backward":
        motor_bwd()
    elif direction == "left":
        left_rot()
    elif direction == "right":
        right_rot()
    return '200'

@app.route('/stop')
def stop_robot():
    print("stop")
    stop()
    return '200'

@app.route('/lights')
def lights():
    on_or_off = request.args.get('on')
    if on_or_off == "true":
        SIREN_THREAD.start()
    else:
        SIREN_THREAD.stop()

@app.route('/servo')
def change_servo():
    servo(int(request.args.get('pos')))
    return '200'


def gen(camera):
    while True:
        frame = camera.get_frame()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

@app.route('/video_feed')
def video_feed():
    return Response(gen(VideoCamera()),
                    mimetype='multipart/x-mixed-replace; boundary=frame')

if __name__ == '__main__':
```

```
    SERVO_POS = 90
    LED_ON = False
    SIREN_THREAD = Thread(target=siren)
    global SIREN_THREAD
    app.run(host="0.0.0.0")
```

camera.py

```
import cv2

class VideoCamera(object):
    def __init__(self):
        # Using OpenCV to capture from device 0.
        self.video = cv2.VideoCapture(0)

    def __del__(self):
        self.video.release()

    def get_frame(self):
        success, image = self.video.read()
        image = cv2.flip(cv2.flip(image, 1), 0)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        encode_param = [int(cv2.IMWRITE_JPEG_QUALITY),50]
        ret, jpeg = cv2.imencode('.jpg', image, encode_param)

        return jpeg.tostring()
```