APPENDICES ON SOURCE CODES

Appendix C

Source code for MJPG Streamer (Uctronics, 2018)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/ioctl.h>
#include <errno.h>
#include <signal.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <getopt.h>
#include <pthread.h>
#include <dlfcn.h>
#include <fcntl.h>
#include <syslog.h>
#include <linux/types.h>          /* for videodev2.h */
#include <linux/videodev2.h>

#include "utils.h"
#include "mjpg_streamer.h"

/* globals */
static globals global;

/******************************************************************************
Description.: Display a help message
Input Value.: argv[0] is the program name and the parameter progname
Return Value: -
******************************************************************************/
static void help(char *progname)
{
   fprintf(stderr, "---------------------------------------------------------------\n");
   fprintf(stderr, "Usage: %s\n" \
         "  -i | --input \"<input-plugin.so> [parameters]\"\n" \
         "  -o | --output \"<output-plugin.so> [parameters]\"\n" \
         " [-h | --help ].......: display this help\n" \
         " [-v | --version ].....: display version information\n" \
         " [-b | --background]...: fork to the background, daemon mode\n", progname);
   fprintf(stderr, "---------------------------------------------------------------\n");
   fprintf(stderr, "Example #1:\n" \
         " To open an UVC webcam \"/dev/video1\" and stream it via HTTP:\n" \
         "  %s -i \"input_uvc.so -d /dev/video1\" -o \"output_http.so\"\n", progname);
   fprintf(stderr, "---------------------------------------------------------------\n");
   fprintf(stderr, "Example #2:\n" \
         " To open an UVC webcam and stream via HTTP port 8090:\n" \
         "  %s -i \"input_uvc.so\" -o \"output_http.so -p 8090\"\n", progname);
   fprintf(stderr, "---------------------------------------------------------------\n");
   fprintf(stderr, "Example #3:\n" \
         " To get help for a certain input plugin:\n" \
         "  %s -i \"input_uvc.so --help\"\n", progname);
   fprintf(stderr, "---------------------------------------------------------------\n");
   fprintf(stderr, "In case the modules (=plugins) can not be found:\n" \
         " * Set the default search path for the modules with:\n" \
```

```
          "   export LD_LIBRARY_PATH=/path/to/plugins,\n" \
          " * or put the plugins into the \"/lib/\" or \"/usr/lib\" folder,\n" \
          " * or instead of just providing the plugin file name, use a complete\n" \
          "   path and filename:\n" \
          "   %s -i \"/path/to/modules/input_uvc.so\"\n", progname);
   fprintf(stderr, "---------------------------------------------------------------------\n");
}

/******************************************************************************
Description.: pressing CTRL+C sends signals to this process instead of just
              killing it plugins can tidily shutdown and free allocated
              resources. The function prototype is defined by the system,
              because it is a callback function.
Input Value.: sig tells us which signal was received
Return Value: -
******************************************************************************/
static void signal_handler(int sig)
{
   int i;

   /* signal "stop" to threads */
   LOG("setting signal to stop\n");
   global.stop = 1;
   usleep(1000 * 1000);

   /* clean up threads */
   LOG("force cancellation of threads and cleanup resources\n");
   for(i = 0; i < global.incnt; i++) {
      global.in[i].stop(i);
      /*for (j = 0; j<MAX_PLUGIN_ARGUMENTS; j++) {
         if (global.in[i].param.argv[j] != NULL) {
            free(global.in[i].param.argv[j]);
         }
      }*/
   }

   for(i = 0; i < global.outcnt; i++) {
      global.out[i].stop(global.out[i].param.id);
      pthread_cond_destroy(&global.in[i].db_update);
      pthread_mutex_destroy(&global.in[i].db);
      /*for (j = 0; j<MAX_PLUGIN_ARGUMENTS; j++) {
         if (global.out[i].param.argv[j] != NULL)
            free(global.out[i].param.argv[j]);
      }*/
   }
   usleep(1000 * 1000);

   /* close handles of input plugins */
   for(i = 0; i < global.incnt; i++) {
      dlclose(global.in[i].handle);
   }

   for(i = 0; i < global.outcnt; i++) {
      int j, skip = 0;
      DBG("about to decrement usage counter for handle of %s, id #%02d, handle: %p\n", \
         global.out[i].plugin, global.out[i].param.id, global.out[i].handle);

      for(j=i+1; j<global.outcnt; j++) {
        if ( global.out[i].handle == global.out[j].handle ) {
```

```
            DBG("handles are pointing to the same destination (%p == %p)\n", global.out[i].handle,
global.out[j].handle);
            skip = 1;
          }
        }
        if ( skip ) {
          continue;
        }

        DBG("closing handle %p\n", global.out[i].handle);

        dlclose(global.out[i].handle);
    }
    DBG("all plugin handles closed\n");

    LOG("done\n");

    closelog();
    exit(0);
    return;
}

static int split_parameters(char *parameter_string, int *argc, char **argv)
{
    int count = 1;
    argv[0] = NULL; // the plugin may set it to 'INPUT_PLUGIN_NAME'
    if(parameter_string != NULL && strlen(parameter_string) != 0) {
        char *arg = NULL, *saveptr = NULL, *token = NULL;

        arg = strdup(parameter_string);

        if(strchr(arg, ' ') != NULL) {
            token = strtok_r(arg, " ", &saveptr);
            if(token != NULL) {
                argv[count] = strdup(token);
                count++;
                while((token = strtok_r(NULL, " ", &saveptr)) != NULL) {
                    argv[count] = strdup(token);
                    count++;
                    if(count >= MAX_PLUGIN_ARGUMENTS) {
                        IPRINT("ERROR: too many arguments to input plugin\n");
                        return 0;
                    }
                }
            }
        }
        free(arg);
    }
    *argc = count;
    return 1;
}

/******************************************************************************
Description.:
Input Value.:
Return Value:
******************************************************************************/
int main(int argc, char *argv[])
{
    //char *input  = "input_uvc.so --resolution 640x480 --fps 5 --device /dev/video0";
```

```
    char *input[MAX_INPUT_PLUGINS];
    char *output[MAX_OUTPUT_PLUGINS];
    int daemon = 0, i, j;
    size_t tmp = 0;

    output[0] = "output_http.so --port 8080";
    global.outcnt = 0;
    global.incnt = 0;

    /* parameter parsing */
    while(1) {
      int c = 0;
      static struct option long_options[] = {
        {"help", no_argument, NULL, 'h'},
        {"input", required_argument, NULL, 'i'},
        {"output", required_argument, NULL, 'o'},
        {"version", no_argument, NULL, 'v'},
        {"background", no_argument, NULL, 'b'},
        {NULL, 0, NULL, 0}
      };

      c = getopt_long(argc, argv, "hi:o:vb", long_options, NULL);

      /* no more options to parse */
      if(c == -1) break;

      switch(c) {
      case 'i':
        input[global.incnt++] = strdup(optarg);
        break;

      case 'o':
        output[global.outcnt++] = strdup(optarg);
        break;

      case 'v':
        printf("MJPG Streamer Version: %s\n",
#ifdef GIT_HASH
        GIT_HASH
#else
        SOURCE_VERSION
#endif
        );
        return 0;
        break;

      case 'b':
        daemon = 1;
        break;

      case 'h': /* fall through */
      default:
        help(argv[0]);
        exit(EXIT_FAILURE);
      }
    }

    openlog("MJPG-streamer ", LOG_PID | LOG_CONS, LOG_USER);
    //openlog("MJPG-streamer ", LOG_PID|LOG_CONS|LOG_PERROR, LOG_USER);
    syslog(LOG_INFO, "starting application");
```

```
   /* fork to the background */
   if(daemon) {
      LOG("enabling daemon mode");
      daemon_mode();
   }

   /* ignore SIGPIPE (send by OS if transmitting to closed TCP sockets) */
   signal(SIGPIPE, SIG_IGN);

   /* register signal handler for <CTRL>+C in order to clean up */
   if(signal(SIGINT, signal_handler) == SIG_ERR) {
      LOG("could not register signal handler\n");
      closelog();

 exit(EXIT_FAILURE);
   }

   /*
    * messages like the following will only be visible on your terminal
    * if not running in daemon mode
    */
#ifdef GIT_HASH
   LOG("MJPG Streamer Version: git rev: %s\n", GIT_HASH);
#else
   LOG("MJPG Streamer Version.: %s\n", SOURCE_VERSION);
#endif

   /* check if at least one output plugin was selected */
   if(global.outcnt == 0) {
      /* no? Then use the default plugin instead */
      global.outcnt = 1;
   }

   /* open input plugin */
   for(i = 0; i < global.incnt; i++) {
      /* this mutex and the conditional variable are used to synchronize access to the global picture
buffer */
      if(pthread_mutex_init(&global.in[i].db, NULL) != 0) {
         LOG("could not initialize mutex variable\n");
         closelog();
         exit(EXIT_FAILURE);
      }
      if(pthread_cond_init(&global.in[i].db_update, NULL) != 0) {
         LOG("could not initialize condition variable\n");
         closelog();
         exit(EXIT_FAILURE);
      }

      tmp = (size_t)(strchr(input[i], ' ') - input[i]);
      global.in[i].stop      = 0;
      global.in[i].context   = NULL;
      global.in[i].buf       = NULL;
      global.in[i].size      = 0;
      global.in[i].plugin = (tmp > 0) ? strndup(input[i], tmp) : strdup(input[i]);
      global.in[i].handle = dlopen(global.in[i].plugin, RTLD_LAZY);
      if(!global.in[i].handle) {
         LOG("ERROR: could not find input plugin\n");
         LOG("       Perhaps you want to adjust the search path with:\n");
         LOG("       # export LD_LIBRARY_PATH=/path/to/plugin/folder\n");
```

```
        LOG("       dlopen: %s\n", dlerror());
        closelog();
        exit(EXIT_FAILURE);
    }
    global.in[i].init = dlsym(global.in[i].handle, "input_init");
    if(global.in[i].init == NULL) {
        LOG("%s\n", dlerror());
        exit(EXIT_FAILURE);
    }
    global.in[i].stop = dlsym(global.in[i].handle, "input_stop");
    if(global.in[i].stop == NULL) {
        LOG("%s\n", dlerror());
        exit(EXIT_FAILURE);
    }
    global.in[i].run = dlsym(global.in[i].handle, "input_run");
    if(global.in[i].run == NULL) {
        LOG("%s\n", dlerror());
        exit(EXIT_FAILURE);
    }
    /* try to find optional command */
    global.in[i].cmd = dlsym(global.in[i].handle, "input_cmd");

    global.in[i].param.parameters = strchr(input[i], ' ');

    for (j = 0; j<MAX_PLUGIN_ARGUMENTS; j++) {
        global.in[i].param.argv[j] = NULL;
    }

    split_parameters(global.in[i].param.parameters, &global.in[i].param.argc, global.in[i].param.argv);
    global.in[i].param.global = &global;
    global.in[i].param.id = i;

    if(global.in[i].init(&global.in[i].param, i)) {
        LOG("input_init() return value signals to exit\n");
        closelog();
        exit(0);
    }
}

/* open output plugin */
for(i = 0; i < global.outcnt; i++) {
    tmp = (size_t)(strchr(output[i], ' ') - output[i]);
    global.out[i].plugin = (tmp > 0) ? strndup(output[i], tmp) : strdup(output[i]);
    global.out[i].handle = dlopen(global.out[i].plugin, RTLD_LAZY);
    if(!global.out[i].handle) {
        LOG("ERROR: could not find output plugin %s\n", global.out[i].plugin);
        LOG("       Perhaps you want to adjust the search path with:\n");
        LOG("       # export LD_LIBRARY_PATH=/path/to/plugin/folder\n");
        LOG("       dlopen: %s\n", dlerror());
        closelog();
        exit(EXIT_FAILURE);
    }
    global.out[i].init = dlsym(global.out[i].handle, "output_init");
    if(global.out[i].init == NULL) {
        LOG("%s\n", dlerror());
        exit(EXIT_FAILURE);
    }
    global.out[i].stop = dlsym(global.out[i].handle, "output_stop");
    if(global.out[i].stop == NULL) {
        LOG("%s\n", dlerror());
```

```
            exit(EXIT_FAILURE);
        }
        global.out[i].run = dlsym(global.out[i].handle, "output_run");
        if(global.out[i].run == NULL) {
            LOG("%s\n", dlerror());
            exit(EXIT_FAILURE);
        }

        /* try to find optional command */
        global.out[i].cmd = dlsym(global.out[i].handle, "output_cmd");

        global.out[i].param.parameters = strchr(output[i], ' ');

        for (j = 0; j<MAX_PLUGIN_ARGUMENTS; j++) {
            global.out[i].param.argv[j] = NULL;
        }
        split_parameters(global.out[i].param.parameters, &global.out[i].param.argc,
global.out[i].param.argv);

        global.out[i].param.global = &global;
        global.out[i].param.id = i;
        if(global.out[i].init(&global.out[i].param, i)) {
            LOG("output_init() return value signals to exit\n");
            closelog();
            exit(EXIT_FAILURE);
        }
    }

    /* start to read the input, push pictures into global buffer */
    DBG("starting %d input plugin\n", global.incnt);
    for(i = 0; i < global.incnt; i++) {
        syslog(LOG_INFO, "starting input plugin %s", global.in[i].plugin);
        if(global.in[i].run(i)) {
            LOG("can not run input plugin %d: %s\n", i, global.in[i].plugin);
            closelog();
            return 1;
        }
    }

    DBG("starting %d output plugin(s)\n", global.outcnt);
    for(i = 0; i < global.outcnt; i++) {
        syslog(LOG_INFO, "starting output plugin: %s (ID: %02d)", global.out[i].plugin,
global.out[i].param.id);
        global.out[i].run(global.out[i].param.id);
    }

    /* wait for signals */
    pause();

    return 0;
}
```

Appendix D

Source code for Motor Driver (Uctronics, 2018)

```
#include <stdio.h>
#include <wiringPi.h>
#include <softPwm.h>
#include <softServo.h>
#include <unistd.h>
#include "motordriver.h"

#define BIT(bit) (1 << (bit))


static uint8_t latch_state; // The state of the shift register on the physical board.

/*
Write to shift register.
*/
void latch_tx(void)
{
  unsigned char i;

  digitalWrite (MOTORLATCH, LOW);

  digitalWrite (MOTORDATA, LOW);

  for (i=0; i<8; i++)
  {
     delayMicroseconds(1);  // 10 micros  delayMicroseconds

    digitalWrite (MOTORCLK, LOW);

    if (latch_state & BIT(7-i))
            {
                    digitalWrite (MOTORDATA, HIGH);
            }
    else
            {
                    digitalWrite (MOTORDATA, LOW);
            }

     delayMicroseconds(1);  // 10 micros  delayMicroseconds
    digitalWrite (MOTORCLK, HIGH);
  }

  digitalWrite (MOTORLATCH, HIGH);
  return;
}

/*
* Initialize GPIOs and shift register.
*/
int ControllerInit(void)
{
            wiringPiSetup () ;

            pinMode (MOTORLATCH,  OUTPUT);
            pinMode(MOTORDATA,   OUTPUT);
            pinMode(MOTORCLK,    OUTPUT);
```

```
            latch_state = 0;
            latch_tx();
            return 0;
}

/*
* Sets everything to LOW and calls gpioTerminate().
*/
void ControllerShutdown(void)
{
   DCMotorRun(1, RELEASE);
   DCMotorRun(2, RELEASE);
   DCMotorRun(3, RELEASE);
   DCMotorRun(4, RELEASE);

 // gpioTerminate();
  return;
}

/*
* Reset a motor?
*/
void DCMotorInit(uint8_t num)
{
  switch (num)
  {
    case 1: latch_state &= ~BIT(MOTOR1_A) & ~BIT(MOTOR1_B);
    case 2: latch_state &= ~BIT(MOTOR2_A) & ~BIT(MOTOR2_B);
    case 3: latch_state &= ~BIT(MOTOR3_A) & ~BIT(MOTOR3_B);
    case 4: latch_state &= ~BIT(MOTOR4_A) & ~BIT(MOTOR4_B);
    default: return;
  }


  latch_tx();

 // printf("Latch=%08X\n", latch_state);
  return;
}

/*
* Change latch_state and write it out to the shift register.
*/
void DCMotorRun(uint8_t motornum, uint8_t cmd)
{
  uint8_t a, b;

  switch (motornum)
  {
    case 1:
                    a = MOTOR1_A;
                    b = MOTOR1_B;
                    break;
    case 2:
                    a = MOTOR2_A;
                    b = MOTOR2_B;
                    break;
    case 3:
                    a = MOTOR3_A;
                    b = MOTOR3_B;
```

```
                        break;
    case 4:
                        a = MOTOR4_A;
                        b = MOTOR4_B;
                        break;
    default: return;
  }

  switch (cmd)
  {
    case FORWARD:
                        latch_state |=  BIT(a);
                        latch_state &= ~BIT(b);
                        break;
    case BACKWARD:
                        latch_state &= ~BIT(a);
                        latch_state |=  BIT(b);
                        break;
    case RELEASE:
                        latch_state &= ~BIT(a);
                        latch_state &= ~BIT(b);
                        break;
    default: return;
  }

  latch_tx();

 // printf("Latch=%08X\n", latch_state);
  return;
}


void go_forward(void) {
            DCMotorRun(1, FORWARD);
            DCMotorRun(2, FORWARD);
            DCMotorRun(3, FORWARD);
            DCMotorRun(4, FORWARD);
return;
}

void go_left(void) {
            DCMotorRun(2, FORWARD);
            DCMotorRun(1, BACKWARD);
            DCMotorRun(3, BACKWARD);
            DCMotorRun(4, FORWARD);
return;
}

void go_right(void) {
            DCMotorRun(2, BACKWARD);
            DCMotorRun(1, FORWARD);
            DCMotorRun(3, FORWARD);
            DCMotorRun(4, BACKWARD);
return;
}

void go_forward_left(void) {
            DCMotorRun(2, FORWARD);
            DCMotorRun(1, RELEASE);
            DCMotorRun(3, RELEASE);
```

```
        DCMotorRun(4, FORWARD);
return;
}

void go_forward_right(void) {
        DCMotorRun(2, RELEASE);
        DCMotorRun(1, FORWARD);
        DCMotorRun(3, FORWARD);
        DCMotorRun(4, RELEASE);
return;
}

void go_back(void) {
        DCMotorRun(1, BACKWARD);
        DCMotorRun(2, BACKWARD);
        DCMotorRun(3, BACKWARD);
        DCMotorRun(4, BACKWARD);
return;
}

void go_back_left(void) {
        DCMotorRun(2, BACKWARD);
        DCMotorRun(1, RELEASE);
        DCMotorRun(3, RELEASE);
        DCMotorRun(4, BACKWARD);
return;
}

void go_back_right(void) {
        DCMotorRun(2, RELEASE);
        DCMotorRun(1, BACKWARD);
        DCMotorRun(3, BACKWARD);
        DCMotorRun(4, RELEASE);
return;
}

void stop(void) {
        DCMotorRun(1, RELEASE);
        DCMotorRun(2, RELEASE);
        DCMotorRun(3, RELEASE);
        DCMotorRun(4, RELEASE);
return;
}
/*
* A test.
*/
void TestRun(void)
{
   DCMotorRun(1, FORWARD);
   sleep(2);
   DCMotorRun(1, RELEASE);

   DCMotorRun(2, FORWARD);
   sleep(2);
   DCMotorRun(2, RELEASE);

   DCMotorRun(3, FORWARD);
   sleep(2);
   DCMotorRun(3, RELEASE);
```

```
   DCMotorRun(4, FORWARD);
   sleep(2);
   DCMotorRun(4, RELEASE);

   sleep(1);
return;
}

/*
* Main

int main (int argc, char *argv[])
{
        if (gpioInitialise() < 0) return 1;

        ControllerInit();

        TestRun();

        ControllerShutdown();

  return 0;
}
*/
```

Appendix E

Source Code for API for Ultrasonic Sensor and Camera (Uctronics, 2018)

```
const app     = require('express')();
const cors    = require('cors');
const helmet  = require('helmet');
const logger  = require('morgan');
const server  = require('http').Server(app);
const io      = require('socket.io')(server);
const netSock = require('net').Socket;
const cSocket = new netSock();

// helmet middleware implementation
app.use(helmet());

// logger middleware implementation
app.use(logger('dev'));

// cors middleware implementation
app.use(cors());

// Car controller commands.
const car = {
  left: 0,
  front: 1,
  right: 2,
  back: 3,
  stopLeft: 4,
  stop: 5,
  stopRight: 6,
  startTracking: 11,
  stopTracking: 12,
  startAutoPilot: 17,
  stopAutoPilot: 18,
  powerOff: 19,
};

// Camera controller commands.
const cam = {
  left: 7,
  right: 8,
  up: 9,
  down: 10,
};

// App Closed.
cSocket.on('close', reason => {
  console,log('Socket Closed with Error?', reason);
});

// Server Error.
cSocket.on('error', error => {
  console.log('Socket Error:', error);
});

//  Car Response.
cSocket.on('data', data => {
  const res = JSON.parse(data.toString() || {});
  console.log('Socket Received:', res);
});
```

```
// Connection to car socket.
cSocket.connect(2001, () => {
  console.log('Socket Connected Successfully');
});

// Entry point.
app.get('/', (req, res) => {
  console.log('Welcome...');
  res.json({
    active: true,
    status: 'OK',
  });
});

// Waiting for users connection.
io.on('connection', (socket) => {
  console.log('A User connected');

  // Client disconnected.
  socket.on('disconnect', () => {
    console.log('A User Disconnected');
  });

  // Front Event.
  socket.on('driveFront', args => {
    const dir = Buffer.from([car.front]);
    cSocket.write(dir, err => {
      console.log('Front');
    });
  });

  // Back Event.
  socket.on('driveBack', args => {
    const dir = Buffer.from([car.back]);
    cSocket.write(dir, err => {
      console.log('Back');
    });
  });

  // Right Event.
  socket.on('driveRight', args => {
    const dir = Buffer.from([car.right]);
    cSocket.write(dir, err => {
      console.log('Right');
    });
  });

  // Left Event.
  socket.on('driveLeft', args => {
    const dir = Buffer.from([car.left]);
    cSocket.write(dir, err => {
      console.log('Left');
    });
  });

  // Stop all directional Events.
  socket.on('driveStop', args => {
    const dir = Buffer.from([car.stop]);
    cSocket.write(dir, err => {
```

```
    console.log('Stop');
  });
});

// Stop Left Event.
socket.on('driveStopLeft', args => {
  const dir = Buffer.from([car.stopLeft]);
  cSocket.write(dir, err => {
    console.log('Stop Left');
  });
});

// Stop Right Event.
socket.on('driveStopRight', args => {
  const dir = Buffer.from([car.stopRight]);
  cSocket.write(dir, err => {
    console.log('Stop Right');
  });
});

// Start Line Tracking Event.
socket.on('driveStartTracking', args => {
  const dir = Buffer.from([car.startTracking]);
  cSocket.write(dir, err => {
    console.log('Tracking Started');
  });
});

// Stop Line Tracking Event.
socket.on('driveStopTracking', args => {
  const dir = Buffer.from([car.stopTracking, car.stop]);
  cSocket.write(dir, err => {
    console.log('Tracking Stopped');
  });
});

// Start Auto Piloting Event.
socket.on('driveStartAutoPilot', args => {
  const dir = Buffer.from([car.startAutoPilot]);
  cSocket.write(dir, err => {
    console.log('Auto Pilot Started');
  });
});

// Stop Auto Pilot Event.
socket.on('driveStopAutoPilot', args => {
  const dir = Buffer.from([car.stopAutoPilot, car.stop]);
  cSocket.write(dir, err => {
    console.log('Auto Pilot Stopped');
  });
});

// Cam Up Event.
socket.on('camUp', args => {
  const dir = Buffer.from([cam.up]);
  cSocket.write(dir, err => {
    console.log('Camera Up');
  });
});
```

```
  // Cam Down Event.
  socket.on('camDown', args => {
    const dir = Buffer.from([cam.down]);
    cSocket.write(dir, err => {
      console.log('Camera Down');
    });
  });

  // Cam Left Event.

  socket.on('camLeft', args => {
    const dir = Buffer.from([cam.left]);
    cSocket.write(dir, err => {
      console.log('Camera Left');
    });
  });

  // Cam Right Event.
  socket.on('camRight', args => {
    const dir = Buffer.from([cam.right]);
    cSocket.write(dir, err => {
      console.log('Camera Right');
    });
  });

  // Power Off Event.
  socket.on('powerOff', args => {
    const dir = Buffer.from([car.powerOff]);
    cSocket.write(dir, err => {
      console.log('Power Off');
    });
  });

});

// server listener.
server.listen(3000, () => {
  console.log('Server Listening On: 3000');
```

Appendix F

Source code for Remote server (Uctronics, 2018)

```
#include <stdio.h>

#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <wiringPi.h>
#include <softPwm.h>
#include <netdb.h>
#include <netinet/in.h>
#include <strings.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "motordriver.h"
#include "remoteserver.h"
#include <pthread.h>
#include <sys/time.h>

#include <signal.h>
#include <stdarg.h>
#include <getopt.h>


#include "clk.h"
#include "gpio.h"
#include "dma.h"
#include "pwm.h"

#include "ws2811.h"

int baseSpeed, addLeftSpeed, addRightSpeed;
unsigned long getColour = 0xFF0000;
unsigned int getBrightness = 100;
static int speedVal_1 = 5000;
static int speedVal_2 = 5000;
static int speedVal_3 = 5000;
static int speedVal_4 = 5000;


struct motionstate carstate = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
static unsigned char disWarning  = 0;
static unsigned char poweroffFlag = 0;
static unsigned char turnLeftFlag = 0;
static unsigned char turnBackFlag = 0;
static unsigned char servoBeep = 0;
unsigned char buf[IR_LIMITS];  // bytes buffer
unsigned int bits;  // 32768 bits capable
unsigned char done;
int sockfd, newsockfd, portno, clilen;
unsigned char client_Connected = 0;
unsigned char count;

#define BLOCK_SIZE (4*1024)

#define ARRAY_SIZE(stuff)       (sizeof(stuff) / sizeof(stuff[0]))
```

```
// defaults for cmdline options
#define TARGET_FREQ           WS2811_TARGET_FREQ
#define GPIO_PIN              18
#define DMA                   10
#define STRIP_TYPE            WS2811_STRIP_RGB    // WS2812/SK6812RGB integrated chip+leds
//#define STRIP_TYPE            WS2811_STRIP_GBR    // WS2812/SK6812RGB integrated chip+leds
//#define STRIP_TYPE            SK6812_STRIP_RGBW   // SK6812RGBW (NOT SK6812RGB)
#define WIDTH                 4
#define HEIGHT                4
#define LED_COUNT             (WIDTH * HEIGHT)

void  INThandler(int sig);
void exit_IKPEZE_Robot(void);
int  mem_fd;
void *gpio_map;
// I/O access
volatile unsigned *gpio;
int width = WIDTH;
int height = HEIGHT;
int led_count = LED_COUNT;
int clear_on_exit = 0;
ws2811_t ledstring =
{
  .freq = TARGET_FREQ,
  .dmanum = DMA,
  .channel =
  {
    [0] =
    {
      .gpionum = GPIO_PIN,
      .count = LED_COUNT,
      .invert = 0,
      .brightness = 255,
      .strip_type = STRIP_TYPE,
    },
    [1] =
    {
      .gpionum = 0,
      .count = 0,
      .invert = 0,
      .brightness = 0,
    },
  },
};

ws2811_led_t *matrix;

unsigned long  grb_colour_table[] =
{
  0xFF0000,  // green
  0x00FF00,  // red
  0x0000FF,  // blue
  0xFFFF00,  //yellow
  0xFF00FF,
  0x00FFFF,  //pink
  0xFFFFFF,  // white
  //0x000000,
};
unsigned long  receive_colour_table[4] =
{
```

```
  0xFF0000,  // green
  0x00FF00,  // red
  0x0000FF,  // blue
  0xFFFF00,  //yellow
};




int PhaseScratchCmd(char command);


/* Creates a server socket and listens for a command from the remote.
*/
int main(int argc, char *argv[])
{
  usleep(10);
  char buffer[BUFFER_SIZE];
  struct sockaddr_in serv_addr, cli_addr;
    struct sigaction sa;
  int  n, pulsenum, count ;
  static unsigned long previous_time = 0;
  static unsigned long now_time = 0;
  static unsigned long time_stamp = 0;
  static int getColIndex = 0;
  static  unsigned char readNowTime = 1;
  signal(SIGINT, INThandler);
  /* Initialise GPIO */
  if (ControllerInit() < 0) return -1;
  setup_io();
  ultraInit();
  servoInit();
  trackModeInit();
  beepInit();
  irInit();
  myPWMInit();
  GRBInit();
  pthread_t t1, t2;
  //creat two thread
  pthread_create(&t1, NULL, fun1, NULL);
  pthread_create(&t2, NULL, fun2, NULL);
  for (pulsenum = 0; pulsenum < 10; pulsenum++) {
    servoCtrl(servo_1, 1140);
    servoCtrl(servo_2, 630);
  }
  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd < 0) {
    perror("ERROR opening socket");
    exit(1);
  }
  /* Initialize socket structure */
  bzero((char *) &serv_addr, sizeof(serv_addr));
  portno = 2001;
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = INADDR_ANY;
  serv_addr.sin_port = htons(portno);

  /* Now bind the host address using bind() call.*/
  if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
```

```
  perror("ERROR on binding");
  exit(1);
}
  /* Add support reconnection */
sa.sa_handler = SIG_IGN;
sigaction( SIGPIPE, &sa, 0 );
/* Now start listening for the clients, here process will
  go in sleep mode and will wait for the incoming connection
*/
while (1) {
  listen(sockfd, 5);
  clilen = sizeof(cli_addr);
  printf("Waitting connection...\r\n");
  client_Connected = 0;
  clearFlag();
  stop();
  BEEP_OPEN();
  /* Accept actual connection from the client */
  newsockfd = accept(sockfd, (struct sockaddr *)&cli_addr, (void *) &clilen);
  if (newsockfd < 0) {
    perror("ERROR on accept");
    //exit(1);
  }
  /* If connection is established then start communicating */
  printf("connect successfully\r\n");

  sleep(0.5);
  GRB_work(3, getColour, getBrightness);
  n = write(newsockfd, "{\"version\":2}", 13);
  client_Connected = 1;
  sleep(0.001);
  bzero(&buffer, BUFFER_SIZE);
  while ((n = read(newsockfd, &buffer, BUFFER_SIZE)) > 0)
  {
    if (buffer[0] == 's') { //0x73
      baseSpeed = buffer[1];
      addLeftSpeed = buffer[2];
      addRightSpeed = buffer[3];
      speedVal_1 = 10000 * ((buffer[1] + buffer[2]) / 255.0);
      speedVal_3 = 10000 * ((buffer[1] + buffer[2]) / 255.0);

      speedVal_2 = 10000 * ((buffer[1] + buffer[3]) / 255.0);
      speedVal_4 = 10000 * ((buffer[1] + buffer[3]) / 255.0);

    } else if (buffer[0] == 'c') { //0x63
      getColour = (buffer[1] << 16) | (buffer[2] << 8) | buffer[3];
      getBrightness = buffer[4];
      GRB_work(3, getColour, getBrightness);
                        if(readNowTime){
                                readNowTime = 0;
                                previous_time = get_pwm_timestamp();
                        }
              now_time = get_pwm_timestamp();
    time_stamp = now_time - previous_time;
    if (time_stamp <2000000) {
                          printf("set current direction\r\n");
                        receive_colour_table[getColIndex] = getColour;
    }else{
                        readNowTime = 1;
                        printf("set next direction\r\n");
```

```
                    getColIndex = (1+getColIndex)%4;
                    receive_colour_table[getColIndex] = getColour;
                    previous_time = get_pwm_timestamp();
            }

    } else if (buffer[0] == 'v') {
      printf("Reveive value %d\n", buffer[0]);
      write(newsockfd, "{\"version\":2}", 13);
    }
    else {
                for(count = 0; count <n; count ++){
                        printf("receive data %d\r\n",buffer[count]);
                }
                if(buffer[0]==0xFF && buffer[1] == 0x55){
                                for (count = 2; count < n; count ++) {
                                        PhaseScratchCmd(buffer[count]);
                                        }
                }else{
                        for (count = 0; count < n; count ++) {
                                        updateCarState(buffer[count]);
                                        updateCarMotion();
                                        }
                }
    }
    bzero(&buffer, BUFFER_SIZE);
   }
   stop();
   clearFlag();
   if (n < 0) {
     printf("Connection exception!\n");
     //break;
   }
 }
 client_Connected = 0;
 close(sockfd);
 printf("ERROR\r\n");
 return 0;
}

void *fun1(void *arg) {
 unsigned char cnt = 0;
 float dis = 0, temp_min = 0, temp_max = 0, temp_value = 0, temp = 0;
 while (1) {
   usleep(1);
   temp_min  = temp_max;
   temp_value = temp_max;
   for (cnt = 0; cnt < 3; cnt ++ ) {
     temp = disMeasure(); usleep(1);
     if (temp_max < temp) temp_max = temp;
     if (temp_min > temp) temp_min = temp;
     temp_value += temp;
   }
   dis = (temp_value - temp_max - temp_min);
   temp_value = 0; temp_min = 0; temp_max = 0; temp_value = 0; temp = 0;
   if (dis > 0 && dis <= 50) {
     disWarning = 1;
     if (carstate.forward) {
       if (!(carstate.autoAvoid)) {
         stop();
       }
```

```
    }
   } else disWarning = 0;

   if (!poweroffFlag) {
    beepWarning();
   }
        if(servoBeep){
                servoBeep = 0;
                digitalWrite(BEEP, HIGH);
                delay(100);
                digitalWrite(BEEP, LOW);
        }


  }
}

void *fun2(void *arg) {
  int IRVal = 0;
  while (1) {
    usleep(1);
    if (carstate.trackenable) {
      printf("Come in track mode \n");
      trackModeWork();
    }
    if (carstate.autoAvoid) {
      avoidance();
    }
    if (done == 1) {
      done = 0;
      IRVal = buf[2]; IRVal = IRVal << 8;
      IRVal = IRVal | buf[3];
      IR_updateCarState(IRVal);
      IR_updateCarMotion();
      IRVal = 0;
    }
    mySoftPwmWrite1(speedVal_1);
    mySoftPwmWrite2(speedVal_2);
    mySoftPwmWrite3(speedVal_3);
    mySoftPwmWrite4(speedVal_4);
    if (!poweroffFlag)  getLedSta();
  }

}

int updateCarMotion(void) {
  static int angleA = 1140;
  static int angleB = 1140;
  char direction[16];
  if (carstate.forward && !carstate.back) {
    if ((!carstate.left && !carstate.right) ||
        (carstate.left && carstate.right)) {
      strcpy(direction, "forward");
      go_forward();
    } else if (carstate.left && !carstate.right) {
      strcpy(direction, "forward left");
      go_forward_left();
    } else if (carstate.right && !carstate.left) {
      strcpy(direction, "forward right");
```

```
    go_forward_right();
  } else {
    strcpy(direction, "stop");
    stop();
  }
}
else if (carstate.back && !carstate.forward) {
  if ((!carstate.left && !carstate.right) ||
      (carstate.left && carstate.right)) {
    strcpy(direction, "back");
    go_back();

  } else if (carstate.left && !carstate.right) {
    strcpy(direction, "back left");
    go_back_left();
  } else if (carstate.right && !carstate.left) {
    strcpy(direction, "back right");
    go_back_right();
  } else {
    strcpy(direction, "stop");
    stop();
  }
} else if (carstate.left && !carstate.right) {
  strcpy(direction, "left");
  go_left();
} else if (carstate.right && !carstate.left) {
  strcpy(direction, "right");
  go_right();
} else if( carstate. stop){
              carstate. stop = 0;
              stop();
}
 if (carstate.servoLeft) {
  carstate.servoLeft = 0;
  strcpy(direction, "servo_left");
  if (angleA < 2000)
    angleA = angleA + 50;
  else{
    angleA = 2000;servoBeep =1;
    //BEEP_OPEN();
           }
           servoCtrl(servo_1,  angleA);
  printf("angleA = %d\r\n", angleA);
}
if (carstate.servoRight) {
  carstate.servoRight = 0;
  strcpy(direction, "servo_right");
  if (angleA > 600)
    angleA = angleA - 50;
  else{
           angleA = 600;servoBeep =1;
           }
 servoCtrl(servo_1,  angleA);
  printf("angleA = %d\r\n", angleA);
}
if (carstate.servoUp) {
  carstate.servoUp = 0;
  strcpy(direction, "servo_UP");
  if (angleB > 600)
    angleB = angleB - 50;
```

```
   else{
                        angleB = 600;servoBeep =1;
            }
            servoCtrl(servo_2,  angleB);

   printf("angleB = %d\r\n", angleB);
  } else if (carstate.servoDown) {
   carstate.servoDown = 0;
   strcpy(direction, "servo_down");
   if (angleB < 2000)
     angleB = angleB + 50;
   else{
                        angleB = 2000;servoBeep =1;
            }
     servoCtrl(servo_2,  angleB);
   printf("angleB = %d\r\n", angleB);
  }
  printf("Motion: %s \n", direction);
  return 0;
}
void  clearFlag(void) {
  carstate.left = 0;
  carstate.forward = 0;
  carstate.right = 0;
  carstate. stop = 0;
  carstate. back = 0;
  carstate.servoLeft = 0;
  carstate. servoRight = 0;
  carstate. servoUp = 0;
  carstate. servoDown = 0;
  carstate. speedUp = 0;
  carstate. speedDown = 0;
  carstate. autoAvoid = 0;

}

int IR_updateCarMotion(void) {
  static int angleA = 1140;
  static int angleB = 630;
  char direction[16];
  if (carstate.forward) {
    strcpy(direction, "forward");
    go_forward(); printf("Motion: %s \n", direction);
  }
  if (carstate.back) {

    strcpy(direction, "back");
    go_back(); printf("Motion: %s \n", direction);

  }
  if (carstate.left) {
    strcpy(direction, "left");
    go_left(); printf("Motion: %s \n", direction);
  }

  if (carstate.right) {
    strcpy(direction, "right");
    go_right(); printf("Motion: %s \n", direction);
  }
  if (carstate.servoLeft) {
```

```
  carstate.servoLeft = 0;
  strcpy(direction, "servo_left");
  if (angleA < 2300)
    angleA = angleA + 50;
  else
    angleA = 2300;
  servoCtrl(servo_1,  angleA);
  printf("angleA = %d\r\n", angleA);
 }
 if (carstate.servoRight) {
  carstate.servoRight = 0;
  strcpy(direction, "servo_right");
  if (angleA > 300)
    angleA = angleA - 50;
  else
    angleA = 300;
  servoCtrl(servo_1,  angleA);
  printf("angleA = %d\r\n", angleA);
 }
 if (carstate.servoUp) {
  carstate.servoUp = 0;
  strcpy(direction, "servo_UP");
  if (angleB > 300)
    angleB = angleB - 50;
  else
    angleB = 300;
  servoCtrl(servo_2,  angleB);
  printf("angleB = %d\r\n", angleB);
 }
 if (carstate.servoDown) {
  carstate.servoDown = 0;
  strcpy(direction, "servo_down");
  if (angleB < 1160)
    angleB = angleB + 50;
  else
    angleB = 1160;
  servoCtrl(servo_2,  angleB);

  printf("angleB = %d\r\n", angleB);
 }
 return 0;
}

int IR_updateCarState(int command) {

 if ( command == IR_up || command == IR_up_v2) {
  if (!disWarning)
    carstate.forward = 1;
  else
    carstate.forward = 0;
 }
 if ( command == IR_down || command == IR_down_v2) {
  carstate.back = 1;
 }
 if ( command == IR_Left || command == IR_Left_v2) {
  carstate.left = 1;
 }
 if ( command == IR_right || command == IR_right_v2) {
  carstate.right = 1;
 }
```

```
 if ( command == IR_stop || command == IR_stop_v2) {
  carstate.forward = 0;
  carstate.back = 0;
  carstate.right = 0;
  carstate.left = 0;
  stop(); printf("Motion: stop \n" );
 }
 if ( command == IR_speed_up) {
  carstate.speedUp = 1;
 }
 if ( command == IR_speed_down) {
  carstate.speedDown = 1;
 }
 if ( command == IR_servo_up) {
  carstate.servoUp = 1;
 }
 if ( command == IR_servo_down) {
  carstate.servoDown = 1;
 }
 if ( command == IR_servo_left) {
  carstate.servoLeft = 1;
 }
 if ( command == IR_servo_right) {
  carstate.servoRight = 1;
 }
 if ( command == IR_track) {
  carstate.trackenable = 1;
 }
 if ( command == IR_track_stop) {
  carstate.trackenable = 0;
  stop();
  printf("Motion: stop \n" );
 }
 return 0;
}


int PhaseScratchCmd(char command){
        static int angleA = 1140;
        static int angleB = 630;

        switch (command) {
   case 1: // go forward
     go_forward();
     GRB_work(3, receive_colour_table[2], getBrightness);
     break;
   case 2: //go backward
     if (!disWarning) {
        go_back();
      GRB_work(3, receive_colour_table[0], getBrightness);
     }
   else
                stop();
          carstate.trackenable = 0;
          carstate.autoAvoid = 0;
     break;
   case 3: //go left
     go_left();
     GRB_work(3, receive_colour_table[3], getBrightness);
     carstate.trackenable = 0;
```

```
    carstate.autoAvoid = 0;
    break;
  case 4: //go right
    go_right();
    GRB_work(3, receive_colour_table[1], getBrightness);
    carstate.trackenable = 0;
    carstate.autoAvoid = 0;
    break;
  case 5: //stop
    stop();
    carstate.trackenable = 0;
    carstate.autoAvoid = 0;
    break;
  case 7: /* servo left */
     if (angleA < 2300)
           angleA = angleA + 50;
             else
            angleA = 2300;
     servoCtrl(servo_1,  angleA);
   break;
  case 8: /*servo right */
    if (angleA > 300)
    angleA = angleA - 50;
   else
    angleA = 300;
   servoCtrl(servo_1,  angleA);
   break;
  case 9: /* servo up */
    if (angleB > 300)
           angleB = angleB - 50;
    else
           angleB = 300;
    servoCtrl(servo_2,  angleB);
  break;
  case 10: /* servo down */
   if (angleB < 1160)
           angleB = angleB + 50;
   else
    angleB = 1160;
   servoCtrl(servo_2,  angleB);
  break;
  case 11: /* enable track */
    carstate.trackenable = 1;
    break;
  case 12: /* disable track */
    carstate.trackenable = 0;
    break;
case 13:
    carstate.speedUp = 1;
    break;
  case 14:
    carstate.speedDown = 1;
    break;
  case 15: /* disable track */
    digitalWrite(BEEP, HIGH);
    break;
  case 16: /* disable track */
    digitalWrite(BEEP, LOW);
    break;
  case 17: /*automatic avoidance*/
```

```
      carstate.autoAvoid = 1;
      break;
    case 18: /*stop automatic avoidance*/
      carstate.autoAvoid = 0;
      break;
    case 19: /*turn off the robot car*/
      poweroffFlag = 1;
      exit_IKPEZE_Robot();
      printf("power off\n");
      system("sudo poweroff");
      break;
  }
  return 0;
}

/* Updates the struct MotionState of the car.
*/
int updateCarState(char command) {
  switch (command) {
    case 0: /* left */
      carstate.left = 1;
      GRB_work(3, receive_colour_table[2], getBrightness);
      break;
    case 1: /* up */
      if (!disWarning) {
        carstate.forward = 1;
        GRB_work(3, receive_colour_table[0], getBrightness);
      }
    else
      carstate.forward = 0;
              carstate.trackenable = 0;
              carstate.autoAvoid = 0;
      break;
    case 2: /* right */
      carstate.right = 1;
      GRB_work(3, receive_colour_table[3], getBrightness);
      carstate.trackenable = 0;
      carstate.autoAvoid = 0;
      break;
    case 3: /* down */
      carstate.back = 1;
      GRB_work(3, receive_colour_table[1], getBrightness);
      carstate.trackenable = 0;
      carstate.autoAvoid = 0;
      break;
    case 4: /* stop left */
      carstate.left = 0;
      carstate.trackenable = 0;
      carstate.autoAvoid = 0;
      break;
    case 5: /* stop*/
      carstate.forward = 0;
      carstate.back = 0;
              carstate.left = 0;
              carstate.right = 0;
      carstate.trackenable = 0;
      carstate.autoAvoid = 0;
              carstate. stop = 1;
      break;
    case 6: /* stop right */
```

```
      carstate.right = 0;
      carstate.trackenable = 0;
      carstate.autoAvoid = 0;
      break;
    case 7: /* servo left */
      carstate.servoLeft = 1;
      break;
    case 8: /*servo right */
      carstate.servoRight = 1;
      break;
    case 9: /* servo up */
      carstate.servoUp = 1;
      break;
    case 10: /* servo down */
      carstate.servoDown = 1;
      break;
    case 11: /* enable track */
      carstate.trackenable = 1;
      break;
    case 12: /* disable track */
      carstate.trackenable = 0;
      break;
    case 13: /* enable track */
      carstate.speedUp = 1;
      break;
    case 14: /* disable track */
      carstate.speedDown = 1;
      break;
    case 15: /* disable track */
      //  carstate.beepenable = 1;
      digitalWrite(BEEP, HIGH);
      break;
    case 16: /* disable track */
      //  carstate.beepenable = 0;
      digitalWrite(BEEP, LOW);
      break;
    case 17: /*automatic avoidance*/
      carstate.autoAvoid = 1;
      break;
    case 18: /*stop automatic avoidance*/
      carstate.autoAvoid = 0;
      break;
    case 19: /*turn off the robot car*/
      poweroffFlag = 1;
      exit_IKPEZE_Robot();
      printf("power off\n");
      system("sudo poweroff");

      break;
  }
  return 0;
}
void ultraInit(void)
{
  pinMode(Echo, INPUT);
  pinMode(Trig, OUTPUT);
}

float disMeasure(void)
{
```

```
  struct timeval tv1;
  struct timeval tv2;
  long start, stop;
  float dis;
  long waitCount = 0;

  digitalWrite(Trig, LOW);
  delayMicroseconds(2);

  digitalWrite(Trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(Trig, LOW);
  waitCount = 0;
  while (!(digitalRead(Echo) == 1)) {
    if (++waitCount >= 5000)
      break;
    else sleep(0.001);
  }
  gettimeofday(&tv1, NULL);
  waitCount = 0;
  while (!(digitalRead(Echo) == 0)) {
    if (++waitCount >= 5000)
      break;
    else sleep(0.001);
  }
  gettimeofday(&tv2, NULL);
  /*
    int gettimeofday(struct timeval *tv, struct timezone *tz);
    The functions gettimeofday() and settimeofday() can get and set the time as well as a timezone.
    The use of the timezone structure is obsolete; the tz argument should normally be specified as
NULL.
  */
  start = tv1.tv_sec * 1000000 + tv1.tv_usec;
  stop  = tv2.tv_sec * 1000000 + tv2.tv_usec;

  dis = (float)(stop - start) / 1000000 * 34000 / 2;

  return dis;
}

void trackModeInit() {
  INP_GPIO(leftSensor);
  GPIO_PULL = 1 << leftSensor;
  INP_GPIO(middleSensor);
  GPIO_PULL = 1 << middleSensor;
  INP_GPIO(rightSensor);
  GPIO_PULL = 1 << rightSensor;


}
void beepInit() {
  pinMode(BEEP, OUTPUT);
  digitalWrite(BEEP, LOW);
}
void getLedSta() {

  static unsigned long previous_time = 0;
  static unsigned long now_time = 0;
  static unsigned long time_stamp = 0;
  static unsigned char flag = 0;
```

```
  static unsigned char colour_index = 0;
  if (!disWarning) {

    if (!client_Connected) {
      if (!flag) {
        flag = 1;
        previous_time = get_pwm_timestamp();
      }
      now_time = get_pwm_timestamp();
      time_stamp = now_time - previous_time;
      if (time_stamp > 500000) {
        time_stamp = 0;
        flag = 0;
        colour_index = (1 + colour_index) % 7;
        //GRB_work(3, grb_colour_table[colour_index], 50);
        GRB_MultiColour_work(3, 100 );
      }
    }
  }
}
void beepWarning() {
  static unsigned long previous_time = 0;
  static unsigned long now_time = 0;
  static unsigned long time_stamp = 0;
  static unsigned char flag = 0;
  static unsigned char canStopBeep = 0;
  if (disWarning) {
    canStopBeep = 1;
    if (!flag) {
      flag = 1;
      previous_time = get_pwm_timestamp();
    }
    now_time = get_pwm_timestamp();
    time_stamp = now_time - previous_time;
    if (time_stamp > 0 && time_stamp <= 50000 ) { //1/2T
      digitalWrite(BEEP, HIGH);
      GRB_work(3, 0, 0);
    }
    if (time_stamp > 50000 && time_stamp <= 2 * 50000 ) { //1T
      digitalWrite(BEEP, LOW);
      GRB_work(3, getColour, getBrightness);
    }
    if (time_stamp > 2 * 50000) {
      flag = 0;
    }
  } else if (canStopBeep) {
    canStopBeep = 0;
    digitalWrite(BEEP, LOW);
    GRB_work(3, getColour, getBrightness);
  }
}

void trackModeWork() {
  int num1 = 0, num2 = 0, num3 = 0;
  while (1) {
    if (!(carstate.trackenable))break;

    mySoftPwmWrite1(speedVal_1);
    mySoftPwmWrite2(speedVal_2);
    mySoftPwmWrite3(speedVal_3);
```

```
  mySoftPwmWrite4(speedVal_4);


  num1 = GET_GPIO(leftSensor);
  num2 = GET_GPIO(middleSensor);
  num3 = GET_GPIO(rightSensor);
  if ((num2 == 0) && (num1 == 0) && (num3 == 0)) {
    stop(); continue;
  } else if ( (num1 == 0) && num3) { //go to right
                                GRB_work(3, grb_colour_table[0], 100 ) ;
    go_forward_left();
    while (1) {
                          GRB_work(3, grb_colour_table[0], 100 ) ;
      num2 = GET_GPIO(middleSensor);
      mySoftPwmWrite1(speedVal_1);
      mySoftPwmWrite2(speedVal_2);
      mySoftPwmWrite3(speedVal_3);
      mySoftPwmWrite4(speedVal_4);
      if (num2) {
        if (!(carstate.trackenable))break;
        if (disWarning) {
          stop();
        }
        else {
                                      GRB_work(3, grb_colour_table[1], 100 ) ;
          go_forward_left();
        }
        continue;
      } else
        break;
    }
  } else if ((num3 == 0) && num1) { // go to left
    go_forward_right();
    while (1) {
      num2 = GET_GPIO(middleSensor);
      mySoftPwmWrite1(speedVal_1);
      mySoftPwmWrite2(speedVal_2);
      mySoftPwmWrite3(speedVal_3);
      mySoftPwmWrite4(speedVal_4);
      if (num2) {
        if (!(carstate.trackenable))break;
        if (disWarning) {
          stop();
        }
        else {
                                      GRB_work(3, grb_colour_table[2], 100 ) ;
          go_forward_right();
        }
        continue;
      } else
        break;
    }
  } else if (disWarning) {
    stop();
  }
  else {
    go_forward();
  }

}
```

```
}

unsigned char countLow(void)
{
  unsigned char i = 0;
  while ( digitalRead(IRIN) == 0 ); // wait
  while ( digitalRead(IRIN) == 1 ) {
    ++i;
    delayMicroseconds(26);
    if (i == 0) return 0; // timeout
  }
  return i;
}

void getIR() {
  unsigned char i;
  unsigned short j;   // capable 32768 bits = 4096 bytes
  unsigned char k;
  bits = 0;
  i = countLow();
  for (j = 0; j < IR_LIMITS; j++) { // buffer bytes LIMITS
    for (i = 0; i < 8; i++) { // 8 bits
      k = countLow();
      if (k == 0) {
        buf[j] >>= (8 - i);
        done = 1; printf("  \n");
        return;
      }
      buf[j] >>= 1;
      buf[j] += ((k > 30) ? 0x80 : 0);
      ++bits;
    }
  }
  done = 1;
}
void turn() {
  static unsigned long previous_time = 0;
  static unsigned long now_time = 0;
  static unsigned long time_stamp = 0;
  static unsigned char flag = 0;
  if (!flag) {
    flag = 1;
    previous_time = get_pwm_timestamp();
  }
  now_time = get_pwm_timestamp();
  time_stamp = now_time - previous_time;
  if (time_stamp > 0 && time_stamp <= turnTime ) { //1/2T
    go_back();turnBackFlag = 1;
  }
  if (time_stamp > turnTime && time_stamp <= 2 * turnTime ) { //1T
    go_left();turnLeftFlag = 1;
  }
  if (time_stamp > 2 * turnTime) {
    stop(); flag = 0;turnLeftFlag = 0;turnBackFlag = 0;
  }
}

void avoidance(void)
{
```

```
  while (1) {
    if (!(carstate.autoAvoid)) {
      stop();
      break;
    }
    mySoftPwmWrite1(speedVal_1);
    mySoftPwmWrite2(speedVal_2);
    mySoftPwmWrite3(speedVal_3);
    mySoftPwmWrite4(speedVal_4);
    if (disWarning || turnLeftFlag||turnBackFlag) {
      turn();
    } else {
      printf("Go forward\n");
      go_forward();
    }
  }
}
void BEEP_INT (void) {
  digitalWrite(BEEP, HIGH);
}

void BEEP_OPEN (void) {
  digitalWrite(BEEP, HIGH);
  delay(500);
  digitalWrite(BEEP, LOW);
}

// Set up a memory regions to access GPIO
void setup_io()
{
  /* open /dev/mem */
  if ((mem_fd = open("/dev/mem", O_RDWR | O_SYNC) ) < 0) {
    printf("can't open /dev/mem \n");
    exit(-1);
  }
  /* mmap GPIO */
  gpio_map = mmap(
              NULL,             //Any adddress in our space will do
              BLOCK_SIZE,       //Map length
              PROT_READ | PROT_WRITE, // Enable reading & writting to mapped memory
              MAP_SHARED,       //Shared with other processes
              mem_fd,           //File to map
              GPIO_BASE         //Offset to GPIO peripheral
            );

  close(mem_fd); //No need to keep mem_fd open after mmap

  if (gpio_map == MAP_FAILED) {
    printf("mmap error %d\n", (int)gpio_map);//errno also set!
    exit(-1);
  }
  // Always use volatile pointer!
  gpio = (volatile unsigned *)gpio_map;
} // setup_io


uint64_t get_pwm_timestamp()
{
  struct timespec t;
```

```
 if (clock_gettime(CLOCK_MONOTONIC_RAW, &t) != 0) {
   return 0;
 }
 return (uint64_t) t.tv_sec * 1000000 + t.tv_nsec / 1000;
}

void mySoftPwmWrite1( int value)
{
 static int previous_time = 0;
 static int now_time = 0;
 static int time_stamp = 0;
 static  unsigned char flag1 = 0;

 if (!flag1) {
   flag1 = 1;
   previous_time = get_pwm_timestamp();
 }
 now_time = get_pwm_timestamp();
 time_stamp = now_time - previous_time;
 if (time_stamp > 0 && time_stamp <= halfPWMPeriod ) { //1/2T
   if (time_stamp <= value ) {
     GPIO_SET = 1 << MOTOR_1_PWM;
   }
   else {
     GPIO_CLR = 1 << MOTOR_1_PWM;
   }
 }
 if (time_stamp > halfPWMPeriod && time_stamp <= 2 * halfPWMPeriod ) { //1T
   if (time_stamp <= value ) {
     GPIO_SET = 1 << MOTOR_1_PWM;
   }
   else {
     GPIO_CLR = 1 << MOTOR_1_PWM;
   }
 }
 if (time_stamp > 2 * halfPWMPeriod) {
   flag1 = 0;
 }
}
void mySoftPwmWrite2( int value)
{
 static int previous_time = 0;
 static int now_time = 0;
 static int time_stamp = 0;
 static  unsigned char flag2 = 0;
 if (!flag2) {
   flag2 = 1;
   previous_time = get_pwm_timestamp();
 }
 now_time = get_pwm_timestamp();
 time_stamp = now_time - previous_time;
 if (time_stamp > 0 && time_stamp <= halfPWMPeriod ) { //1/2T
   if (time_stamp <= value ) {
     GPIO_SET = 1 << MOTOR_2_PWM;
   }
   else {
     GPIO_CLR = 1 << MOTOR_2_PWM;
   }
 }
 if (time_stamp > halfPWMPeriod && time_stamp <= 2 * halfPWMPeriod ) { //1T
```

```
    if (time_stamp <= value ) {
      GPIO_SET = 1 << MOTOR_2_PWM;
    }
    else {
      GPIO_CLR = 1 << MOTOR_2_PWM;
    }
  }
  if (time_stamp > 2 * halfPWMPeriod) {
    flag2 = 0;
  }
}
void mySoftPwmWrite3( int value)
{
  static int previous_time = 0;
  static int now_time = 0;
  static int time_stamp = 0;
  static  unsigned char flag3 = 0;
  if (!flag3) {
    flag3 = 1;
    previous_time = get_pwm_timestamp();
  }
  now_time = get_pwm_timestamp();
  time_stamp = now_time - previous_time;
  if (time_stamp > 0 && time_stamp <= halfPWMPeriod ) { //1/2T
    if (time_stamp <= value ) {
      GPIO_SET = 1 << MOTOR_3_PWM;
    }
    else {
      GPIO_CLR = 1 << MOTOR_3_PWM;
    }
  }
  if (time_stamp > halfPWMPeriod && time_stamp <= 2 * halfPWMPeriod ) { //1T
    if (time_stamp <= value ) {
      GPIO_SET = 1 << MOTOR_3_PWM;
    }
    else {
      GPIO_CLR = 1 << MOTOR_3_PWM;
    }
  }
  if (time_stamp > 2 * halfPWMPeriod) {
    flag3 = 0;
  }
}
void mySoftPwmWrite4( int value)
{
  static int previous_time = 0;
  static int now_time = 0;
  static int time_stamp = 0;
  static  unsigned char flag4 = 0;

  if (!flag4) {
    flag4 = 1;
    previous_time = get_pwm_timestamp();
  }
  now_time = get_pwm_timestamp();
  time_stamp = now_time - previous_time;
  if (time_stamp > 0 && time_stamp <= halfPWMPeriod ) { //1/2T
    if (time_stamp <= value ) {
      GPIO_SET = 1 << MOTOR_4_PWM;
    }
```

```
   else {
     GPIO_CLR = 1 << MOTOR_4_PWM;
   }
  }
  if (time_stamp > halfPWMPeriod && time_stamp <= 2 * halfPWMPeriod ) { //1T
   if (time_stamp <= value ) {
     GPIO_SET = 1 << MOTOR_4_PWM;
   }
   else {
     GPIO_CLR = 1 << MOTOR_4_PWM;
   }
  }
  if (time_stamp > 2 * halfPWMPeriod) {
   flag4 = 0;
  }
}

void servoInit(void)
{
  INP_GPIO(servo_1); // must use INP_GPIO before we can use OUT_GPIO
  OUT_GPIO(servo_1);
  INP_GPIO(servo_2); // must use INP_GPIO before we can use OUT_GPIO
  OUT_GPIO(servo_2);
}

void servoCtrl(int servoNum, int dutyCycle)
{
        int count =0;
        for(count = 0; count < 3; count ++){
                GPIO_SET = 1 << servoNum;
                delayMicroseconds(dutyCycle);
                GPIO_CLR = 1 << servoNum;
                delayMicroseconds(25000 - dutyCycle);
        }
}

void servoAControl( int value)
{
  static int previous_time = 0;
  static int now_time = 0;
  static int time_stamp = 0;
  static  unsigned char flag1 = 0;

  if (!flag1) {
   flag1 = 1;
   previous_time = get_pwm_timestamp();
f  }
  now_time = get_pwm_timestamp();
  time_stamp = now_time - previous_time;
  if (time_stamp > 0 && time_stamp <= 10000 ) { //1/2T
   if (time_stamp <= value ) {
     GPIO_SET = 1 << servo_1;
   }
   else {
     GPIO_CLR = 1 << servo_1;
   }
  }
  if (time_stamp > 10000 && time_stamp <= 2 * 10000 ) { //1T
   if (time_stamp <= value ) {
     GPIO_SET = 1 << servo_1;
```

```
    }
    else {
      GPIO_CLR = 1 << servo_1;
    }
  }
  if (time_stamp > 2 * 10000) {
    flag1 = 0;
  }
}
void GRB_work(unsigned int ledNum, unsigned long colour, int brightness ) {
  int i;
  ledstring.channel[0].brightness = brightness;
  for (i = 0; i < ledNum; i++) {
    ledstring.channel[0].leds[i] = colour;
  }
  ws2811_render(&ledstring) ;

}

void GRB_MultiColour_work(unsigned int ledNum,  int brightness ) {
  int i;
  static int colour_index = 0;

  ledstring.channel[0].brightness = brightness;
  for (i = 0; i < ledNum; i++) {
    colour_index = (1 + colour_index) % 7;
    ledstring.channel[0].leds[i] = grb_colour_table[colour_index];
  }
  ws2811_render(&ledstring) ;

}


void irInit() {
  pinMode(IRIN, INPUT);
  pullUpDnControl(IRIN, PUD_UP);
  // to detect falling edge
  wiringPiISR(IRIN, INT_EDGE_FALLING, (void*)getIR);
  done = 0;
}
void myPWMInit() {
  // Switch GPIO 7..11 to output mode
  INP_GPIO(MOTOR_1_PWM); // must use INP_GPIO before we can use OUT_GPIO
  OUT_GPIO(MOTOR_1_PWM);
  INP_GPIO(MOTOR_2_PWM); // must use INP_GPIO before we can use OUT_GPIO
  OUT_GPIO(MOTOR_2_PWM);
  INP_GPIO(MOTOR_3_PWM); // must use INP_GPIO before we can use OUT_GPIO
  OUT_GPIO(MOTOR_3_PWM);
  INP_GPIO(MOTOR_4_PWM); // must use INP_GPIO before we can use OUT_GPIO
  OUT_GPIO(MOTOR_4_PWM);
}
void GRBInit() {
  ws2811_return_t ret;
  matrix = malloc(sizeof(ws2811_led_t) * width * height);
  if ((ret = ws2811_init(&ledstring)) != WS2811_SUCCESS)
  {
    fprintf(stderr, "ws2811_init failed: %s\n", ws2811_get_return_t_str(ret));
  }

}
```

```
void exit_IKPEZE_Robot(void)
{
  int pulsenum;
  client_Connected = 0;
  close(sockfd);
  for (pulsenum = 0; pulsenum < 10; pulsenum++) {
    servoCtrl(servo_1, 1300);
    servoCtrl(servo_2, 1300);
  }
  digitalWrite(BEEP, LOW);
  GRB_work(3, 0, 0 );
}

void  INThandler(int sig)
{
  char  c;
  signal(sig, SIG_IGN);
  printf("You hit Ctrl-C\n"
         "Do you really want to quit? [y/n] ");
  c = getchar();
  if (c == 'y' || c == 'Y') {
    exit_IKPEZE_Robot();
    printf("Bye.\n");
    exit(0);
  }

  else
    signal(SIGINT, INThandler);
  getchar(); // Get new line character
}
```

Appendix G

Source Code for Mobile App in Ionic Framework

```
package com.ionicframework.cordova.webview;

import android.app.Activity;

import android.content.SharedPreferences;

import org.apache.cordova.CallbackContext;

import org.apache.cordova.CordovaPlugin;

import org.json.JSONArray;

import org.json.JSONException;

public class IonicWebView extends CordovaPlugin  {

 public static final String WEBVIEW_PREFS_NAME = "WebViewSettings";

 public static final String CDV_SERVER_PATH = "serverBasePath";

 public boolean execute(String action, JSONArray args, CallbackContext callbackContext) throws
JSONException {

    if (action.equals("setServerBasePath")) {

      final String path = args.getString(0);

      cordova.getActivity().runOnUiThread(new Runnable() {

       public void run() {

          ((IonicWebViewEngine)webView.getEngine()).setServerBasePath(path);

        }

      });

      return true;

    } else if (action.equals("getServerBasePath")) {

      callbackContext.success(((IonicWebViewEngine)webView.getEngine()).getServerBasePath());

      return true;

    } else if (action.equals("persistServerBasePath")) {

      String path = ((IonicWebViewEngine)webView.getEngine()).getServerBasePath();

      SharedPreferences prefs =
cordova.getActivity().getApplicationContext().getSharedPreferences(WEBVIEW_PREFS_NAME,
Activity.MODE_PRIVATE);

      SharedPreferences.Editor editor = prefs.edit();

      editor.putString(CDV_SERVER_PATH, path);
```

```
      editor.apply();
      return true;
    }
    return false;


package com.ionicframework.cordova.webview;
import android.app.Activity;
import android.content.Context;
import android.content.SharedPreferences;
import android.content.pm.PackageInfo;
import android.annotation.TargetApi;
import android.graphics.Bitmap;
import android.net.Uri;
import android.os.Build;
import android.support.annotation.RequiresApi;
import android.util.Log;
import android.webkit.WebResourceRequest;
import android.webkit.WebResourceResponse;
import android.webkit.WebSettings;
import android.webkit.WebView;
import org.apache.cordova.ConfigXmlParser;
import org.apache.cordova.CordovaInterface;
import org.apache.cordova.CordovaPreferences;
import org.apache.cordova.CordovaResourceApi;
import org.apache.cordova.CordovaWebView;
import org.apache.cordova.CordovaWebViewEngine;
import org.apache.cordova.NativeToJsMessageQueue;
import org.apache.cordova.PluginManager;
import org.apache.cordova.engine.SystemWebViewClient;
import org.apache.cordova.engine.SystemWebViewEngine;
import org.apache.cordova.engine.SystemWebView;
public class IonicWebViewEngine extends SystemWebViewEngine {
  public static final String TAG = "IonicWebViewEngine";
```

```
  private WebViewLocalServer localServer;
  private String CDV_LOCAL_SERVER;
  private String scheme;
  private static final String LAST_BINARY_VERSION_CODE = "lastBinaryVersionCode";
  private static final String LAST_BINARY_VERSION_NAME = "lastBinaryVersionName";
  /**
   * Used when created via reflection.
   */
  public IonicWebViewEngine(Context context, CordovaPreferences preferences) {
    super(new SystemWebView(context), preferences);
    Log.d(TAG, "Ionic Web View Engine Starting Right Up 1...");
  }
  public IonicWebViewEngine(SystemWebView webView) {
    super(webView, null);
    Log.d(TAG, "Ionic Web View Engine Starting Right Up 2...");
  }
  public IonicWebViewEngine(SystemWebView webView, CordovaPreferences preferences) {
    super(webView, preferences);
    Log.d(TAG, "Ionic Web View Engine Starting Right Up 3...");
  }
  @Override
  public void init(CordovaWebView parentWebView, CordovaInterface cordova, final
CordovaWebViewEngine.Client client,
                   CordovaResourceApi resourceApi, PluginManager pluginManager,
                   NativeToJsMessageQueue nativeToJsMessageQueue) {
    ConfigXmlParser parser = new ConfigXmlParser();
    parser.parse(cordova.getActivity());
    String hostname = preferences.getString("Hostname", "localhost");
    scheme = preferences.getString("Scheme", "http");
    CDV_LOCAL_SERVER = scheme + "://" + hostname;
    localServer = new WebViewLocalServer(cordova.getActivity(), hostname, true, parser, scheme);
    localServer.hostAssets("www");
    webView.setWebViewClient(new ServerClient(this, parser));
```

```
    super.init(parentWebView, cordova, client, resourceApi, pluginManager, nativeToJsMessageQueue);
    if (android.os.Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
      final WebSettings settings = webView.getSettings();
      int mode = preferences.getInteger("MixedContentMode", 0);
      settings.setMixedContentMode(mode);
    }
    SharedPreferences prefs =
cordova.getActivity().getApplicationContext().getSharedPreferences(IonicWebView.WEBVIEW_PRE
FS_NAME, Activity.MODE_PRIVATE);
    String path = prefs.getString(IonicWebView.CDV_SERVER_PATH, null);
    if (!isDeployDisabled() && !isNewBinary() && path != null && !path.isEmpty()) {
      setServerBasePath(path);
    }
  }
  private boolean isNewBinary() {
    String versionCode = "";
    String versionName = "";
    SharedPreferences prefs =
cordova.getActivity().getApplicationContext().getSharedPreferences(IonicWebView.WEBVIEW_PRE
FS_NAME, Activity.MODE_PRIVATE);
    String lastVersionCode = prefs.getString(LAST_BINARY_VERSION_CODE, null);
    String lastVersionName = prefs.getString(LAST_BINARY_VERSION_NAME, null);

    try {
      PackageInfo pInfo =
this.cordova.getActivity().getPackageManager().getPackageInfo(this.cordova.getActivity().getPackage
Name(), 0);
      versionCode = Integer.toString(pInfo.versionCode);
      versionName = pInfo.versionName;
    } catch(Exception ex) {
      Log.e(TAG, "Unable to get package info", ex);
    }
    if (!versionCode.equals(lastVersionCode) || !versionName.equals(lastVersionName)) {
      SharedPreferences.Editor editor = prefs.edit();
      editor.putString(LAST_BINARY_VERSION_CODE, versionCode);
```

```
    editor.putString(LAST_BINARY_VERSION_NAME, versionName);
    editor.putString(IonicWebView.CDV_SERVER_PATH, "");
    editor.apply();
    return true;
  }
  return false;
 }
 private boolean isDeployDisabled() {
  return preferences.getBoolean("DisableDeploy", false);
 }
 private class ServerClient extends SystemWebViewClient {
  private ConfigXmlParser parser;

  public ServerClient(SystemWebViewEngine parentEngine, ConfigXmlParser parser) {
    super(parentEngine);
    this.parser = parser;
  }
  @RequiresApi(Build.VERSION_CODES.LOLLIPOP)
  @Override
  public WebResourceResponse shouldInterceptRequest(WebView view, WebResourceRequest
request) {
    return localServer.shouldInterceptRequest(request.getUrl(), request);
  }
  @TargetApi(Build.VERSION_CODES.KITKAT)
  @Override
  public WebResourceResponse shouldInterceptRequest(WebView view, String url) {
    return localServer.shouldInterceptRequest(Uri.parse(url), null);
  }
  @Override
  public void onPageStarted(WebView view, String url, Bitmap favicon) {
    super.onPageStarted(view, url, favicon);
    String launchUrl = parser.getLaunchUrl();
```

```
    if (!launchUrl.contains(WebViewLocalServer.httpsScheme) &&
!launchUrl.contains(WebViewLocalServer.httpScheme) && url.equals(launchUrl)) {

      view.stopLoading();

      // When using a custom scheme the app won't load if server start url doesn't end in /

      String startUrl = CDV_LOCAL_SERVER;

      if (!scheme.equalsIgnoreCase(WebViewLocalServer.httpsScheme) &&
!scheme.equalsIgnoreCase(WebViewLocalServer.httpScheme)) {

        startUrl += "/";

      }

      view.loadUrl(startUrl);

    }

  }

  @Override

  public void onPageFinished(WebView view, String url) {

    super.onPageFinished(view, url);

    view.loadUrl("javascript:(function() { " +

          "window.WEBVIEW_SERVER_URL = '" + CDV_LOCAL_SERVER + "';" +

          "})()");

  }

 }

 public void setServerBasePath(String path) {

  localServer.hostFiles(path);

  webView.loadUrl(CDV_LOCAL_SERVER);

 }

 public String getServerBasePath() {

  return this.localServer.getBasePath();

 }

}
```

Appendix H

Source Code for Android Protocol Handler

```
package com.ionicframework.cordova.webview;
import android.content.Context;
import android.content.res.AssetManager;
import android.net.Uri;
import android.util.Log;
import android.util.TypedValue;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.List;
public class AndroidProtocolHandler {
  private static final String TAG = "AndroidProtocolHandler";

  private Context context;

  public AndroidProtocolHandler(Context context) {
    this.context = context;
  }
  public InputStream openAsset(String path) throws IOException {
    return context.getAssets().open(path, AssetManager.ACCESS_STREAMING);
  }
  public InputStream openResource(Uri uri) {
    assert uri.getPath() != null;
    // The path must be of the form ".../asset_type/asset_name.ext".
    List<String> pathSegments = uri.getPathSegments();
    String assetType = pathSegments.get(pathSegments.size() - 2);
    String assetName = pathSegments.get(pathSegments.size() - 1);
```

```
 // Drop the file extension.
 assetName = assetName.split("\\.")[0];
 try {
  // Use the application context for resolving the resource package name so that we do
  // not use the browser's own resources. Note that if 'context' here belongs to the
  // test suite, it does not have a separate application context. In that case we use
  // the original context object directly.
  if (context.getApplicationContext() != null) {
   context = context.getApplicationContext();
  }
  int fieldId = getFieldId(context, assetType, assetName);
  int valueType = getValueType(context, fieldId);
  if (valueType == TypedValue.TYPE_STRING) {
   return context.getResources().openRawResource(fieldId);
  } else {
   Log.e(TAG, "Asset not of type string: " + uri);
   return null;
  }
 } catch (ClassNotFoundException e) {
  Log.e(TAG, "Unable to open resource URL: " + uri, e);
  return null;
 } catch (NoSuchFieldException e) {
  Log.e(TAG, "Unable to open resource URL: " + uri, e);
  return null;
 } catch (IllegalAccessException e) {
  Log.e(TAG, "Unable to open resource URL: " + uri, e);
  return null;
 }
}
public InputStream openFile(String filePath) throws IOException  {
 String realPath = filePath.replace(WebViewLocalServer.fileStart, "");
 File localFile = new File(realPath);
 return new FileInputStream(localFile);
```

```
  }

  public InputStream openContentUrl(Uri uri)  throws IOException {

    Integer port = uri.getPort();

    String realPath;

    if (port == -1) {

      realPath = uri.toString().replace(uri.getScheme() + "://" + uri.getHost() +
WebViewLocalServer.contentStart, "content:/");

    } else {

      realPath = uri.toString().replace(uri.getScheme() + "://" + uri.getHost() + ":" + port +
WebViewLocalServer.contentStart, "content:/");

    }

    InputStream stream = null;

    try {

      stream = context.getContentResolver().openInputStream(Uri.parse(realPath));

    } catch (SecurityException e) {

      Log.e(TAG, "Unable to open content URL: " + uri, e);

    }

    return stream;

  }

  private static int getFieldId(Context context, String assetType, String assetName)

    throws ClassNotFoundException, NoSuchFieldException, IllegalAccessException {

    Class<?> d = context.getClassLoader()

      .loadClass(context.getPackageName() + ".R$" + assetType);

    java.lang.reflect.Field field = d.getField(assetName);

    int id = field.getInt(null);

    return id;

  }

  private static int getValueType(Context context, int fieldId) {

    TypedValue value = new TypedValue();

    context.getResources().getValue(fieldId, value, true);

    return value.type;

  }

}
```

Appendix I

Source Code for Web Server on Raspberry Pi (Uctronics, 2018)

```
package com.ionicframework.cordova.webview;

import android.content.Context;

import android.net.Uri;

import android.os.Build;

import android.util.Log;

import android.webkit.WebResourceRequest;

import android.webkit.WebResourceResponse;

import org.apache.cordova.ConfigXmlParser;

import java.io.IOException;

import java.io.InputStream;

import java.net.HttpURLConnection;

import java.net.SocketTimeoutException;

import java.net.URL;

import java.net.URLConnection;

import java.util.HashMap;

import java.util.Map;

import java.util.UUID;

/**

 * Helper class meant to be used with the android.webkit.WebView class to enable hosting assets,

 * resources and other data on 'virtual' http(s):// URL.

 * Hosting assets and resources on http(s):// URLs is desirable as it is compatible with the

 * Same-Origin policy.

 * <p>

 * This class is intended to be used from within the

 * {@link android.webkit.WebViewClient#shouldInterceptRequest(android.webkit.WebView, String)}
and

 * {@link android.webkit.WebViewClient#shouldInterceptRequest(android.webkit.WebView,

 * android.webkit.WebResourceRequest)}

 * methods.

 */
```

```
public class WebViewLocalServer {
  private static String TAG = "WebViewAssetServer";
  private String basePath;
  public final static String httpScheme = "http";
  public final static String httpsScheme = "https";
  public final static String fileStart = "/_app_file_";
  public final static String contentStart = "/_app_content_";
  private final UriMatcher uriMatcher;
  private final AndroidProtocolHandler protocolHandler;
  private final String authority;
  private final String customScheme;
  // Whether we're serving local files or proxying (for example, when doing livereload on a
  // non-local endpoint (will be false in that case)
  private boolean isAsset;
  // Whether to route all requests to paths without extensions back to `index.html`
  private final boolean html5mode;
  private ConfigXmlParser parser;
  public String getAuthority() { return authority; }
  /**
   * A handler that produces responses for paths on the virtual asset server.
   * <p>
   * Methods for this handler will be invoked on a background thread and care must be taken to
   * correctly synchronize access to any shared state.
   * <p>
   * On Android KitKat and above these methods may be called on more than one thread. This thread
   * may be different than the thread on which the shouldInterceptRequest method was invoke.
   * This means that on Android KitKat and above it is possible to block in this method without
   * blocking other resources from loading. The number of threads used to parallelize loading
   * is an internal implementation detail of the WebView and may change between updates which
   * means that the amount of time spend blocking in this method should be kept to an absolute
   * minimum.
   */
  public abstract static class PathHandler {
```

```
protected String mimeType;
private String encoding;
private String charset;
private int statusCode;
private String reasonPhrase;
private Map<String, String> responseHeaders;
public PathHandler() {
  this(null, null, 200, "OK", null);
}
public PathHandler(String encoding, String charset, int statusCode,
                String reasonPhrase, Map<String, String> responseHeaders) {
  this.encoding = encoding;
  this.charset = charset;
  this.statusCode = statusCode;
  this.reasonPhrase = reasonPhrase;
  Map<String, String> tempResponseHeaders;
  if (responseHeaders == null) {
    tempResponseHeaders = new HashMap<String, String>();
  } else {
    tempResponseHeaders = responseHeaders;
  }
  tempResponseHeaders.put("Cache-Control", "no-cache");
  this.responseHeaders = tempResponseHeaders;
}
abstract public InputStream handle(Uri url);
public String getEncoding() {
  return encoding;
}
public String getCharset() {
  return charset;
}
public int getStatusCode() {
  return statusCode;
```

```
  }
  public String getReasonPhrase() {
    return reasonPhrase;
  }
  public Map<String, String> getResponseHeaders() {
    return responseHeaders;
  }
}
/**
 * Information about the URLs used to host the assets in the WebView.
 */
public static class AssetHostingDetails {
  private Uri httpPrefix;
  private Uri httpsPrefix;
  /*package*/ AssetHostingDetails(Uri httpPrefix, Uri httpsPrefix) {
    this.httpPrefix = httpPrefix;
    this.httpsPrefix = httpsPrefix;
  }
  /**
   * Gets the http: scheme prefix at which assets are hosted.
   *
   * @return the http: scheme prefix at which assets are hosted. Can return null.
   */
  public Uri getHttpPrefix() {
    return httpPrefix;
  }
  /**
   * Gets the https: scheme prefix at which assets are hosted.
   *
   * @return the https: scheme prefix at which assets are hosted. Can return null.
   */
  public Uri getHttpsPrefix() {
    return httpsPrefix;
```

```
    }
  }
  WebViewLocalServer(Context context, String authority, boolean html5mode, ConfigXmlParser
parser, String customScheme) {
    uriMatcher = new UriMatcher(null);
    this.html5mode = html5mode;
    this.parser = parser;
    this.protocolHandler = new AndroidProtocolHandler(context.getApplicationContext());
    this.authority = authority;
    this.customScheme = customScheme;
  }
  private static Uri parseAndVerifyUrl(String url) {
    if (url == null) {
      return null;
    }
    Uri uri = Uri.parse(url);
    if (uri == null) {
      Log.e(TAG, "Malformed URL: " + url);
      return null;
    }
    String path = uri.getPath();
    if (path == null || path.length() == 0) {
      Log.e(TAG, "URL does not have a path: " + url);
      return null;
    }
    return uri;
  }
  private static WebResourceResponse createWebResourceResponse(String mimeType, String
encoding, int statusCode, String reasonPhrase, Map<String, String> responseHeaders, InputStream
data) {
    if (android.os.Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
      int finalStatusCode = statusCode;
      try {
        if (data.available() == 0) {
```

```
        finalStatusCode = 404;
       }
     } catch (IOException e) {
       finalStatusCode = 500;
     }
     return new WebResourceResponse(mimeType, encoding, finalStatusCode, reasonPhrase,
responseHeaders, data);
   } else {
     return new WebResourceResponse(mimeType, encoding, data);
   }
 }
 /**
  * Attempt to retrieve the WebResourceResponse associated with the given <code>request</code>.
  * This method should be invoked from within
  * {@link android.webkit.WebViewClient#shouldInterceptRequest(android.webkit.WebView,
  * android.webkit.WebResourceRequest)}.
  *
  * @param uri the request Uri to process.
  * @return a response if the request URL had a matching handler, null if no handler was found.
  */
 public WebResourceResponse shouldInterceptRequest(Uri uri, WebResourceRequest request) {
   PathHandler handler;
   synchronized (uriMatcher) {
     handler = (PathHandler) uriMatcher.match(uri);
   }
   if (handler == null) {
     return null;
   }
   if (isLocalFile(uri) || uri.getAuthority().equals(this.authority)) {
     Log.d("SERVER", "Handling local request: " + uri.toString());
     return handleLocalRequest(uri, handler, request);
   } else {
     return handleProxyRequest(uri, handler);
```

```
    }

  }

  private boolean isLocalFile(Uri uri) {

    String path = uri.getPath();

    if (path.startsWith(contentStart) || path.startsWith(fileStart)) {

      return true;

    }

    return false;

  }

  private WebResourceResponse handleLocalRequest(Uri uri, PathHandler handler,
WebResourceRequest request) {

    String path = uri.getPath();

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP && request != null &&
request.getRequestHeaders().get("Range") != null) {

      InputStream responseStream = new LollipopLazyInputStream(handler, uri);

      String mimeType = getMimeType(path, responseStream);

      Map<String, String> tempResponseHeaders = handler.getResponseHeaders();

      int statusCode = 206;

      try {

        int totalRange = responseStream.available();

        String rangeString = request.getRequestHeaders().get("Range");

        String[] parts = rangeString.split("=");

        String[] streamParts = parts[1].split("-");

        String fromRange = streamParts[0];

        int range = totalRange-1;

        if (streamParts.length > 1) {

          range = Integer.parseInt(streamParts[1]);

        }

        tempResponseHeaders.put("Accept-Ranges", "bytes");

        tempResponseHeaders.put("Content-Range", "bytes " + fromRange + "-" + range + "/" +
totalRange);

      } catch (IOException e) {

        statusCode = 404;

      }
```

```
      return createWebResourceResponse(mimeType, handler.getEncoding(),
              statusCode, handler.getReasonPhrase(), tempResponseHeaders, responseStream);
    }
    if (isLocalFile(uri)) {
      InputStream responseStream = new LollipopLazyInputStream(handler, uri);
      String mimeType = getMimeType(path, responseStream);
      return createWebResourceResponse(mimeType, handler.getEncoding(),
              handler.getStatusCode(), handler.getReasonPhrase(), handler.getResponseHeaders(),
responseStream);
    }
    if (path.equals("") || path.equals("/") || (!uri.getLastPathSegment().contains(".") && html5mode)) {
      InputStream stream;
      String launchURL = parser.getLaunchUrl();
      String launchFile = launchURL.substring(launchURL.lastIndexOf("/") + 1, launchURL.length());
      try {
        String startPath = this.basePath + "/" + launchFile;
        if (isAsset) {
          stream = protocolHandler.openAsset(startPath);
        } else {
          stream = protocolHandler.openFile(startPath);
        }
      } catch (IOException e) {
        Log.e(TAG, "Unable to open " + launchFile, e);
        return null;
         return createWebResourceResponse("text/html", handler.getEncoding(),
                handler.getStatusCode(), handler.getReasonPhrase(), handler.getResponseHeaders(), stream);
    }
    int periodIndex = path.lastIndexOf(".");
    if (periodIndex >= 0) {
      InputStream responseStream = new LollipopLazyInputStream(handler, uri);
      String mimeType = getMimeType(path, responseStream);
      return createWebResourceResponse(mimeType, handler.getEncoding(),
```

```
            handler.getStatusCode(), handler.getReasonPhrase(), handler.getResponseHeaders(),
responseStream);
    }
    return null;
  }
  /**
   * Instead of reading files from the filesystem/assets, proxy through to the URL
   * and let an external server handle it.
   * @param uri
   * @param handler
   * @return
   */
  private WebResourceResponse handleProxyRequest(Uri uri, PathHandler handler) {
    try {
      String path = uri.getPath();
      URL url = new URL(uri.toString());
      HttpURLConnection conn = (HttpURLConnection) url.openConnection();
      conn.setRequestMethod("GET");
      conn.setReadTimeout(30 * 1000);
      conn.setConnectTimeout(30 * 1000);
      InputStream stream = conn.getInputStream();
      if (path.equals("/") || (!uri.getLastPathSegment().contains(".") && html5mode)) {
        return createWebResourceResponse("text/html", handler.getEncoding(),
                handler.getStatusCode(), handler.getReasonPhrase(), handler.getResponseHeaders(),
stream);
      }
      int periodIndex = path.lastIndexOf(".");
      if (periodIndex >= 0) {
        String ext = path.substring(path.lastIndexOf("."), path.length());
        // TODO: Conjure up a bit more subtlety than this
        if (ext.equals(".html")) {
        }
        String mimeType = getMimeType(path, stream);
```

```
      return createWebResourceResponse(mimeType, handler.getEncoding(),
          handler.getStatusCode(), handler.getReasonPhrase(), handler.getResponseHeaders(), stream);
    }
    return createWebResourceResponse("", handler.getEncoding(),
          handler.getStatusCode(), handler.getReasonPhrase(), handler.getResponseHeaders(), stream);
  } catch (SocketTimeoutException ex) {
    // bridge.handleAppUrlLoadError(ex);
  } catch (Exception ex) {
    // bridge.handleAppUrlLoadError(ex);
  }
  return null;
}
private String getMimeType(String path, InputStream stream) {
  String mimeType = null;
  try {
    mimeType = URLConnection.guessContentTypeFromName(path); // Does not recognize *.js
    if (mimeType != null && path.endsWith(".js") && mimeType.equals("image/x-icon")) {
      Log.d(IonicWebViewEngine.TAG, "We shouldn't be here");
    }
    if (mimeType == null) {
      if (path.endsWith(".js") || path.endsWith(".mjs")) {
        // Make sure JS files get the proper mimetype to support ES modules
        mimeType = "application/javascript";
      } else if (path.endsWith(".wasm")) {
        mimeType = "application/wasm";
      } else {
        mimeType = URLConnection.guessContentTypeFromStream(stream);
      }
    }
  } catch (Exception ex) {
    Log.e(TAG, "Unable to get mime type" + path, ex);
  }
  return mimeType;
```

```
}
/**
 * Registers a handler for the given <code>uri</code>. The <code>handler</code> will be invoked
 * every time the <code>shouldInterceptRequest</code> method of the instance is called with
 * a matching <code>uri</code>.
 *
 * @param uri     the uri to use the handler for. The scheme and authority (domain) will be matched
 *                exactly. The path may contain a '*' element which will match a single element of
 *                a path (so a handler registered for /a/* will be invoked for /a/b and /a/c.html
 *                but not for /a/b/b) or the '**' element which will match any number of path
 *                elements.
 * @param handler the handler to use for the uri.
 */
void register(Uri uri, PathHandler handler) {
  synchronized (uriMatcher) {
    uriMatcher.addURI(uri.getScheme(), uri.getAuthority(), uri.getPath(), handler);
  }
}
/**
 * Hosts the application's assets on an http(s):// URL. Assets from the local path
 * <code>assetPath/...</code> will be available under
 * <code>http(s)://{uuid}.androidplatform.net/assets/...</code>.
 *
 * @param assetPath the local path in the application's asset folder which will be made
 *                  available by the server (for example "/www").
 */
public void hostAssets(String assetPath) {
  hostAssets(authority, assetPath);
}
/**
 * Hosts the application's assets on an http(s):// URL. Assets from the local path
 * <code>assetPath/...</code> will be available under
 * <code>http(s)://{domain}/{virtualAssetPath}/...</code>.
```

```
   *
   * @param domain           custom domain on which the assets should be hosted (for example
"example.com").
   * @param assetPath        the local path in the application's asset folder which will be made
   *                         available by the server (for example "/www").
   * @return prefixes under which the assets are hosted.
   */
  public void hostAssets(final String domain,
                         final String assetPath) {
    this.isAsset = true;
    this.basePath = assetPath;
    createHostingDetails();
  }
  private void createHostingDetails() {
    final String assetPath = this.basePath;
    if (assetPath.indexOf('*') != -1) {
      throw new IllegalArgumentException("assetPath cannot contain the '*' character.");
    }
    PathHandler handler = new PathHandler() {
      @Override
      public InputStream handle(Uri url) {
        InputStream stream = null;
        String path = url.getPath();
        try {
          if (path.startsWith(contentStart)) {
            stream = protocolHandler.openContentUrl(url);
          } else if (path.startsWith(fileStart) || !isAsset) {
            if (!path.startsWith(fileStart)) {
              path = basePath + url.getPath();
            }
            stream = protocolHandler.openFile(path);
          } else {
            stream = protocolHandler.openAsset(assetPath + path);
```

```
        }
      } catch (IOException e) {
        Log.e(TAG, "Unable to open asset URL: " + url);
        return null;
      }
      return stream;
    }
  };
  registerUriForScheme(httpScheme, handler, authority);
  registerUriForScheme(httpsScheme, handler, authority);
  if (!customScheme.equals(httpScheme) && !customScheme.equals(httpsScheme)) {
    registerUriForScheme(customScheme, handler, authority);
  }
}
private void registerUriForScheme(String scheme, PathHandler handler, String authority) {
  Uri.Builder uriBuilder = new Uri.Builder();
  uriBuilder.scheme(scheme);
  uriBuilder.authority(authority);
  uriBuilder.path("");
  Uri uriPrefix = uriBuilder.build();
  register(Uri.withAppendedPath(uriPrefix, "/"), handler);
  register(Uri.withAppendedPath(uriPrefix, "**"), handler);
}
```