KF5011 Assignment

Dr Alun Moon

Semester 2 — 2017/18

Contents

List of Tables

Part I
Assignment Details and Administration

1 Key Dates and Submission Mechanism

Table 1: Key Dates for assignment

Submission Deadline	23:59 Sunday 13th May 2018
Demonstrations and Marking	Monday 14th May to Friday 18th May
	Location and timetable to be confirmed ***Make sure*** *that it compiles and executes*

1.1 Submission mechanism

The submission mechanism is via Github Classrooms and Blackboard.

There are separate assignment parts, each a software project:

Step 1. Follow the links in table 2, these should give you a private Github repository that you and I have access to.

Step 2. Use the repository for your work, remember to commit and push code regularly (see appendix B).

Step 3. Submit the URL through the Blackboard assignment(s) (Paste URL into the *Write Submission* text box)

Table 2: Assignment URLs

Game Controller	`https://classroom.github.com/a/VRh30nkL` `https://github.com/kf5011/game-controller-seed`
Game Dashboard	`https://classroom.github.com/a/APDfS3Bb` `https://github.com/kf5011/game-dashboard-seed`

1.2 Academic Integrity Statement

1.2.0.1 This is an Individual Assignment. You must adhere to the university regulations on academic conduct. Formal inquiry proceedings will be instigated if there is any suspicion of plagiarism or any other form of misconduct in your work. Refer to the Universitys Assessment Regulations for Northumbria Awards if you are unclear as to the meaning of these terms. The latest copy is available on the University website.

Part II
Assignment Tasks

2 Overview

You are to write programs as described in 2.2 and 2.3. The details are in sections 3, 4. You will be provided with a server that models the flight dynamics and propulsion, this is communicated with via UDP using the protocol described in appendix A.1

2.1 Lunar Lander

In the classic Lunar lander games the player controls the decent of a Lunar Lander spacecraft onto the surface of the moon. [1] [2]

2.2 Game Controller

You are to write a program (C++/Mbed) for the K64F platform and application shield to implement a game controller. This takes input from the player to control the game. It can display some limited information on the display and indicate states using the LEDs.

2.3 Game Dashboard

You are provided with a limited interface on the server, mainly for monitoring the server and debugging. The display on the applications board is black and white with a limited size. To provide a better user interface you are to write a dashboard program that runs on a PC and displays information communicated to it from the K64F.

[1] `https://en.wikipedia.org/wiki/Lunar_Lander_(video_game_genre)`
[2] `https://en.wikipedia.org/wiki/Lunar_Lander_(1979_video_game)`

3 Game Controller

3.1 Requirements

3.1.1 Inputs

Using the sensors and input devices on the FRDM+K64F and Applications shield, monitor the user's activity as inputs to the game. You can use any combination of Accelerometer, buttons, magnetometer, and temperature sensors as you wish.

3.1.2 Outputs

Some feedback to the user can be given using the LEDs, the LCD display, and the speaker. Examples could include

contact light indicates that the lander has touched down successfully

proximity warning to indicate you are close to the ground

fuel warning for when your fuel is running out

3.1.3 Communications

The K64F needs to manage it's communications with the Lander server, the dashboard, and the data logging service. Communications with the Lander server are by UDP Datagrams, the format of the messages and responses are given in appendix A.1. Communication is Synchronous, the server only sends messages in response to requests.

3.1.4 Control Logic

The K64F needs to take the requested actions from the user, these it needs to generate the control signals it needs to communicate to the server. It also needs to monitor the system and present information to the user, on the state of the game etc.

3.2 Programming Hints

Here are some hints[3] and guidance on how to write a good, robust solution.

3.2.0.1 Use a good layer model for hardware. Having functions that provide a logical view of a device (such as `ispressed()`) aid in writing the program and hiding implementation details (such as the different wiring of the switches).

3.2.0.2 Clean Modular Separation of Function. Using separate functions, even source files, or libraries. Keeps functional requirements separate and allows you concentrate on each bit of functionality without the clutter of the other parts.

3.2.0.3 Use Threads, Events, and EventQueues You'll need several threads to handle some of the concurrent tasks, and allow for waits on the IO.

3.3 Workplan

Here is a suggested work-plan to follow

3.3.1 Define State

You will need a set of variables to record the user input, the actions it corresponds to and the returned information from the Lander.

3.3.2 User display

Some of the values defined as states need to be shown to the user. Some can be written to the LCD, others can shown as LEDs on or off. For debugging more detailed output can be written to the Serial Monitor.

This is a lengthy operations and can be done inside a `while` loop within `main`. You may want to call `wait`

[3]Programming hints are gives as code fragments. Each may have a `main` function, you'll have onnly one `main`. They may be other code missing that is assumed, such as preparing and decoding messages.

3.3.3 Lander communications

You will need a periodic task to communicate with the Lander model. You will have to hardwire the IP address of the lander into your code, and it will have to change for each computer you use.

The function should have the prototype

```
void function(void){

}
```

It will be called by the RTOS once for each time period, *it should exit*

To call it at a defined rate use.

```
periodic.call_every( period , function);
```

With the time period given in milli-seconds.

3.3.3.1 Lander To communicate with the Lander:

1. format a message in a buffer, use `sprintf`

   ```
   char buffer[512];
   sprintf(buffer,"command:!\n....");
   ```

2. Send the message to the Lander

   ```
   udp.sendto( lander, buffer, strlen(buffer));
   ```

   remember to use `strlen` here to only send the bytes that make up the message.

3. Get a reply from the Lander

   ```
   nsapi_size_or_error_t  n = udp.recvfrom(&source, buffer, sizeof(buffer));
   buffer[n] = '\0';
   ```

- Remember that the *address-of* operator (`&`) is used for the first parameter.
- Here use `sizeof(buffer)` to set the maximum size of message to receive.
- The `recvfrom` function returns the number of bytes received, *It does not write a zero byte at the end of the data*. You will have to write the zero byte to terminate the string yourself.

4. Parse (decode) the incoming message.

   - Split the text into lines, and handle each line

     ```
     char *nextline, *line;
     for(
         line = strtok_r(buffer, "\r\n", &nextline);
         line != NULL;
         line = strtok_r(NULL, "\r\n", &nextline)
     ) {
     ```

   - each line can be split into a key:value pair

     ```
     char *key, *value;
     key = strtok(line, ":");
     value = strtok(NULL, ":");
     ```

   - There isn't a simple string comparison in C. To test the key string

     ```
     if( strcmp(key,"altitude")==0 ) {
     /* do something with the 'altitude' value */
     }
     ```

     - use `atof` to convert string to `float`
     - use `atoi` to convert string to `int`
     - `bool` can be converted from an `int` 0 or 1

3.3.4 User input

You will need a second task to monitor the user input. Some devices, such as the accelerometer will have to be polled in a periodic task as above. Others such as the switches can either be polled or use interrupts.

3.3.4.1 The Accelerometer can be read using

```
motion_data_units_t a;
acc.getAxis(a);
```

Be cautious the way the accelerometer is mounted you measure the sideways roll in the opposite direction to the lander. You may want to use $-angle$ to make decisions on.

Angle of board If you are using the accelerometer, you may want to know the angle the board is held at. The accelerometer measures values in g, so the range of values in each of the x, y, z axis should vary between -1 and $+1$. In practice they may read a little more due to noise and the sensitivity of the accelerometer and how steady your hand is.

Simple version The simple version of identifying the angle is to use the `a.x` component of the reading directly.

More accurate version To get a more accurate reading of the board angle there are two steps.

1. make sure the *magnitude* of the reading is 1

   ```
   float magnitude = sqrt( a.x*a.x + a.x*a.x + a.x*a.x );
   a.x = a.x/magnitude;
   a.y = a.y/magnitude;
   a.y = a.y/magnitude;
   ```

2. The the angle the board is tilted over at is given by

   ```
   float angle = asin(a.x);
   ```

   Where the `angle` is in radians.

Deadband You may find it difficult to hold the board absolutely level, and that the lander slowly rotates.

To solve this apply a *deadband*, where you ignore readings close to zero.

3.3.4.2 The Buttons It is possible to use the buttons as inputs. These can be polled, or fire interrupts.

Actions Some suggested actions to perform for various button presses are:

joystick left	roll left at a rate of 1 (roll is -1)
joystick right	roll right at a rate of 1 (roll is +1)
joystick down	apply full throttle (100)

Remember That for the joystick buttons on the shield, if the pin reads 1 the button is presssed. For the two switched on the MBED board a 0 is read when the switch is pressed.

3.3.4.3 Potentiometers One posible use for the potentiometers is as a throttle. The potentioments returns a reading between 0 and 1, the throttle should be set to a value between 0 and 100.

4 Dashboard

The dashboard is able to provide a more detailed user interface than can be provided for on the LCD display.

4.1 Requirements

The Dashboard should run on a PC or other computer connected to the K64F via the off-campus network (assuming you are in a Pandon Lab). There are some options for the communications, you are free to choose which ever works for you.

UDP Datagrams (**recommended**) UDP Datagrams will work once you have a network between the two. There is an overhead in that you'll have to hardwire IP addresses into the code and recompile it for different IP address combinations.

PC Serial (**not recommended!**) Using the serial connection via `Serial pc(USBTX, USBRX);`. The downside of this, is that this is the channel used by the serial monitor, so you'd loose that for debugging.

USB Serial Device The connection created by the `USBSerial comm;` library uses the other USB connection on the K64F board and is seen by the PC as a second serial device, this can be used in parallel with the Serial Monitor.

4.2 Asynchronous Communications

The Dashboard need only receive messages from the K64F. You may have ideas on features to add that does need communications in return – feel free to do so.

4.3 Protocol

You will need to use a Protocol that describes the format of the data as sent in the UDP Datagrams. If you end up writing the Dashboard in Java, or Python, then as the MBED/K64F code is in C++, there is a danger of miscommunicating. A protocol allows you to describe the format of the data, independently from the programming languages, and you'll have a better chance of success.

4.3.0.1 The Protocol will also state the Port Number that the Dashboard listens on. The choice is up to you, and you'll need to make sure that the Dashboard listens on the right port, and the controller sends to that port on the PC's IP address.

Part III
Appendicies

A Lander Server

The lander Server is a Java program that models the dynamics of the lander and the lunar surface. It is packaged as a `jar` file and executed using

```
{bash}
java -jar Lander.jar
```

A.1 Communications Protocol

The server listens for UDP Packets on port 20769

A.1.1 Message Format

Messages to the Server and replies are formatted as Key:Value pairs as is done in email headers (RFC822[4]) or HTTP Headers (RFC7230[5]).

Query messages have the message name as the first key, with a single question mark (ascii 63) as the value

Command messages that set values or cause actions, have the message name as the first key and a corresponding exclamation mark (ascii 33) as the value.

Reply messages returned in responce to the above, have the message name as the key and as equals sign (ascii 61) as the value.

A.1.1.1 Command Engines This message sets the requested levels on the engines

[4] `https://tools.ietf.org/html/rfc822`
[5] `https://tools.ietf.org/html/rfc7230`

Message send

```
command:!
throttle: 100
roll:  +0.5
```

The keys and values are:

`throttle`	percentage throttle setting	0...100
`roll`	the strength of the rotational thrust +ve is anti-clockwise	-1...1,

Either key-value pair may be used as a single item in the message, or both used in the same message.

Reply from server

```
command:=
altitude: <altitude as float>
fuel: <percent fuel remaining as float>
flying: <isflying state 0/1>
crashed: <iscrashed 0/1>
orientation: <orientation of lander in degrees>
Vx: <Velocity in x direction>
Vy: <Velocity in y direction (positive is down)>
```

The Lander can be obtained from:

Hesabu `http://hesabu.net/kf5011/Lander.jar`

B Git usage guide

Once you have a clone of the assignment repository you can follow the workflow below. These are shell commands and work for the Linux, Mac OSX, and Windows (Command line) versions of git.

1. Clone the repository – create a local working copy
   `git clone https://github.com/your-git-id/repository-name.git`
   or if you prefer the ssh version
   `git clone git@github.com:your-git-id/repository-name.git`
2. Work on your code...
3. Check which files have changed and/or been added
   `git status`
4. Add the files that you want to keep the changes for
   `git add file1 file2.cpp etc`
   These files will be uploaded back to the github server below
5. Commit the changes, add a note explaining the work you have done
   `git commit -m"message goes here in quotes"`
6. Push the changes back to the github server
   `git push`

Whether you use

Github desktop `https://desktop.github.com/`

GitKraken `https://www.gitkraken.com/`

Atom's Github integration `https://github.atom.io/`

The basic workflow is the same, the details of each interface and front end vary.