Kilobot Simulator Setup Guide

Authors: Dr. Michael Otte & Joseph Katakam

Simulator Intro:

The Kilobot simulator is a program that emulates the behavior of a swarm of Kilobots, and it can be run directly on a web browser. Since web browsers primarily use JavaScript for executing programs, we need to use a tool called Emscripten to compile C code into a format that JavaScript can call. This guide aims to demonstrate how to run the Kilobot simulator on a web browser, by allowing us to write C code as we would normally do and have each robot in the swarm execute that code. Essentially, this involves using Emscripten to convert the C code into a compatible format that can be executed in a browser environment, so that each robot in the swarm can interpret and execute the instructions provided.

I. Emscripten:

Emscripten enables developers to bring existing C/C++ codebases to the web without having to rewrite them in JavaScript. WebAssembly (often abbreviated as WASM) is a low-level, binary format for executing code on the web. It is designed to run in all modern web browsers and provides a portable, efficient, and safe way to run code on the web. WebAssembly is not tied to any specific programming language and can be used with any language that can be compiled into its binary format. More info on Emscripten can be found *here* and *here*.

The Emscripten setup instructions can be found *here*, which have been copied here for convenience.

Step 1: Get the 'emsdk' repo. The core Emscripten SDK (emsdk) driver is a Python script.

- Open WSL- Ubuntu 18.04 and run the below commands
  - `git clone https://github.com/emscripten-core/emsdk.git`

Step 2: Enter that directory

  - `cd emsdk`

Step 3: Fetch the latest version of the emsdk (not needed the first time you clone)

  - `git pull`

Step 4: Download and install the latest SDK tools.

  - `./emsdk install latest`

Step 5: Make the "latest" SDK "active" for the current user. (writes .emscripten file)

- `./emsdk activate latest`

Step 6: Activate PATH and other environment variables in the current terminal

- `source ./emsdk_env.sh`

Notes:

1. git pull will fetch the current list of tags, but very recent ones may not yet be present there.
2. You can run emsdk update tags to update the list of tags directly.
3. If you change the location of the SDK (e.g. take it to another computer on a USB),
4. re-run the emsdk activate latest and emsdk_env.bat commands.

Updating the SDK *(if some time has passed since the installation)*:

Step 1: Fetch the latest registry of available tools.

- `./emsdk update`

Step 2: Download and install the latest SDK tools.

- `./emsdk install latest`

Step 3: Set up the compiler configuration to point to the "latest" SDK.

- `./emsdk activate latest`

Step 4: Activate PATH and other environment variables in the current terminal

- `source ./emsdk_env.sh`

Notes:

- If you get the error: 'emcc' is not recognized as an internal or external command, operable program, or batch file.
- Run the `source ./emsdk_env.sh` command from the emsdk directory.

```
joseph@DarkAngel:~/test$ emcc hello_world.c

Command 'emcc' not found, but can be installed with:

sudo apt install emscripten

joseph@DarkAngel:~/test$ cd
joseph@DarkAngel:~$ cd emsdk/
joseph@DarkAngel:~/emsdk$ source ./emsdk_env.sh
Setting up EMSDK environment (suppress these messages with EMSDK_QUIET=1)
Adding directories to PATH:
PATH += /home/joseph/emsdk
PATH += /home/joseph/emsdk/upstream/emscripten
PATH += /home/joseph/emsdk/node/14.18.2_64bit/bin

Setting environment variables:
PATH = /home/joseph/emsdk:/home/joseph/emsdk/upstream/emscripten:/home/joseph/emsdk/node/14.18.2_64bit/bin:/usr/local/sb
in:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/usr/lib/wsl/lib:/mnt/a/WinAVR/bin:/mnt/a/Wi
nAVR/utils/bin:/mnt/c/Windows/system32:/mnt/c/Windows:/mnt/c/Windows/System32/Wbem:/mnt/c/Windows/System32/WindowsPowerS
hell/v1.0/:/mnt/c/Windows/System32/OpenSSH/:/mnt/c/Program Files/dotnet/:/mnt/c/Program Files (x86)/NVIDIA Corporation/P
hysX/Common:/mnt/c/Program Files/NVIDIA Corporation/NVIDIA NvDLISR:/mnt/c/Program Files/Microsoft VS Code/bin:/mnt/a/Git
/Git/cmd:/mnt/c/Program Files/usbipd-win/:/mnt/c/Users/jpran/AppData/Local/Microsoft/WindowsApps:/snap/bin
EMSDK = /home/joseph/emsdk
EMSDK_NODE = /home/joseph/emsdk/node/14.18.2_64bit/bin/node
```

Pre-requisites Setup:

Step 1: Install Python

- `sudo apt-get install python3`

Step 2: Install CMake

- `sudo apt-get install cmake`

Using Emscripten:

Step 1: Create a test C program

- Our first quest is to craft a magical message that says hello to the whole wide world! 🌎
- Save as “hello_world.c”

```
/*
 * Copyright 2011 The Emscripten Authors.  All rights reserved.
 * Emscripten is available under two separate licenses, the MIT
license and the
 * University of Illinois/NCSA Open Source License.  Both these
licenses can be
 * found in the LICENSE file.
 */

#include <stdio.h>

int main() {
  printf("hello, world!\n");
  return 0;
}
```

Step 2: Go to the directory where “hello_world.c” it exits

- `cd [wherever]`

```
joseph@DarkAngel:~$ cd test
joseph@DarkAngel:~/test$ ls
hello_world.c
```

Step 3: To build the JavaScript version of this code, simply specify the C/C++ file after emcc (use em++ to force compilation as C++):

- Open PowerShell and run the command

- `emcc hello_world.c`

```
joseph@DarkAngel:~/test$ emcc hello_world.c
joseph@DarkAngel:~/test$ ls
a.out.js  a.out.wasm  hello_world.c
```

Step 4: Run the JavaScript

- You should see two files generated by that command: a.out.js and a.out.wasm. The second is a WebAssembly file containing the compiled code, and the first is a JavaScript file containing the runtime support to load and execute it.
- You can run them using node.js:
  - `node a.out.js`

```
joseph@DarkAngel:~/test$ node a.out.js
hello, world!
```

Step 4: Generating HTML

- Emscripten can also generate HTML for testing embedded JavaScript.
- To generate HTML, use the -o (output) command and specify an HTML file as the target file:
  - `emcc hello_world.c -o hello.html`

```
joseph@DarkAngel:~/test$ emcc hello_world.c -o hello.html
joseph@DarkAngel:~/test$ ls
a.out.js  a.out.wasm  hello.html  hello.js  hello.wasm  hello_world.c
```

Step 4: Start a local web server in the directory

- depending on your configuration, you may need to use python2 or python3.
  - `python3 -m http.server`

```
joseph@DarkAngel:~/test$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Step 4: Browser Display

- Open any browser and paste the below link into your browser:
  - http://localhost:8000/hello.html

Quick start for working on Code:

Step 1: Start python webserver

- In one terminal, make sure to do it from the "code for kilobot simulator" directory

```
python3 -m http.server 8080
```

Step 2: Compile code

- In another terminal, from the emsdk directory

```
source ./emsdk_env.sh
```

Step 3: start webpage

- make sure to do it from the "code for kilobot simulator" directory where main.cpp is

```
python3 -m http.server 8080
```

Step 4: in a google chrome browser:

- http://localhost:8090/index.html