Developing for WeBot

Steps:

1. Acquiring libraries
2. Building necessary Wicron libraries
3. Programming on your computer
4. Deploying on the robot
5. Building on the robot

1. Acquiring libraries

When you start to develop things for Webot, the very first thing you need to do is to build the libraries that wicron has provided. They can be found on bitbucket: https://bitbucket.org/wicron/

The basic library is VCORE, you have to build it first because all the other libraries depend on this one. Based on the task you are willing to accomplish, you might also want to build VUNIROBOT or VROBOT. You might also contact Dmitry Suvorov or Roman Zhukov for advice (their contact information is provided below).

Acquiring them is pretty easy if you know **git** — just copy the link and use it in **git clone**. However, If you don't know git, consult google or some of the following links to be more confident using it:
https://git-scm.com/book/en/v2/Getting-Started-Git-Basics (formal)
http://rogerdudler.github.io/git-guide/ (informal)

2. Building necessary Wicron libraries

You basically need to follow the instruction provided on the page with VCORE: https://bitbucket.org/wicron/vcore. Though it should be multiplatform, the guide that follows will cover building this library in Ubuntu. It's better to use Ubuntu, because the robot also has UNIX installed and it would be easier to program in Ubuntu because it would be easier to deploy this code on the robot then.

You have to have BOOST, libmodbus and hidapi. Just follow the instruction on the webpage and you should be safe. The only point i want to make is that I didn't build HIDAPI, I've just set a variable for the path to its sources that I have downloaded in CMAKE and VCORE built and worked alright.

The other libraries you might need to build again depend on VCORE, so be careful with that and make sure you have built VCORE first.

3. Programming on your computer

If you're doing this on Ubuntu (which I strongly recommend) or other Linux-like system, I advice to use QtCreator. Create a Non-QT CMake project because whatever you have to be built on robot will be built with CMake, so it's good to have some practice before deploying it.

Now, if your task is somehow related to robot motors control, I suggest using the class that was provided to control robot's motion, it's called VRobotMotionFacadeImpl. It can be downloaded from this page (https://github.com/morgan1189/skoltech-swarmfleye/tree/master/kinect-webot/) This class has the following methods implemented:

```
bool readParameters( const std::string& vendorDir,
                     int motor_left,
                     int motor_right,
                     int motor_left_reverse,
                     int motor_left_mode,
                     int motor_right_reverse,
                     int motor_right_mode,
                     std::string motor_left_config,
                     std::string motor_right_config,
                     float distance_between_wheels,
                     float wheel_radius,
                     float maxSpeed,
                     float maxOmega );
```

This one is used to configure the motor parameters of the robot.

```
int connectToDevices();
```

This one is used to access the board which is on "/dev/unirobot" and controls all the motors. It returns 0 if the connection was established and 1 if for some reason that wasn't possible.

```
bool setSpeed(int velocity, int omega);
```

This one allows you to set speed of the robot (in percentages of the maximum). So, by calling, for instance robotMotion->setSpeed(50, 20) the robot now is going forward at 50 percent of its maximum speed and rotating at 20 percent of its maximum rotational speed. To make it move or rotate in the other direction, use negative parameters (for instance, setSpeed(-50, -20)).

These are the basic functions you will need to control the robot. You have to note a couple of things here:

1) For safety reasons robot stops moving once every 200 ms, so calling a bunch of setSpeed functions in a row won't work. You have to call std::this_thread::sleep_for(std::chrono::milliseconds(200)) after every setSpeed to avoid this behavior and repeat your setSpeed command if necessary.

2) There is a configuration directory “/vendor” which is needed as vendorDir parameter to the readParameters method.

To test how the libraries work create the project with just VRobotMotionFacadeImpl and main.cpp with the following code:

```
VRobotMotionFacadeImpl * robotMotion = new VRobotMotionFacadeImpl;
  robotMotion->readParameters("/vendor", 1, 0, 0, 0, 1, 0, "IG32E-139K.xml",
"IG32E-139K.xml", 0.346, 0.101, 0.6, 1.6);
  robotMotion->connectToDevices();

  robotMotion->setSpeed(50, 0);
  std::this_thread::sleep_for(std::chrono::milliseconds(200));
  robotMotion->setSpeed(50, 0);
  std::this_thread::sleep_for(std::chrono::milliseconds(200));
  robotMotion->setSpeed(50, 0);
```

This would have made robot move forward for some time but as we’re running it on the machine, we should get the error about “/dev/unirobot” being offline:
ERROR Can't open the device /dev/unirobot (No such file or directory)

It is perfectly fine. If it builds and runs - you have built and used everything successfully and are ready to deploy your program on the robot.

4. Deploying on the robot

Because of the fact that robot has an ARM7 processor, you want to build everything on the robot. However, if you have a binary that has been built on the robot, you should be able to just spread it to other robots.

You would have to manipulate the robot’s filesystem (copying and getting files from it) as well just do some commands in its terminal. The simplest way to do this is SSH:

1) Establish a network on the second adapter of the robot (it’s usually titled Reserve and should be 2.4 GHz).
2) Connect to this network
3) If robot’s user-friendly UI is enabled, you have to get to the desktop. Press the “:|” button in the upper-right corner of the screen, go to Settings, press the “:|” button again and then “Go to OS”.
4) Open the terminal from the Desktop.
5) Issue “ifconfig” in robot’s terminal. Remember the robot’s IP.
6) Use “ssh linaro@<robot_ip>” to connect to the robot via SSH. The password should be provided to you by Dmitry, but at the moment of writing it is: “armv7robot)”.

5. Building on the robot

A few steps before you can build on the robot:

1) Copy the headers for vcore, vunirobot to “/usr/include” and boost to “/usr/include/arm-linux-gnueabihf”. The easiest way to do it is:
   “scp -r <path_to_folder_source> linaro@<robot_ip>:<path_to_folder_destination>”.
   Generally use SCP to copy files to and from the robot.
   The other libraries should already be there.
2) Rename the “boost” folder in “/usr/local/include” to “boost_1.49” (or any name other than just “boost”): “mv boost boost_1.49”. This way you can avoid CMake searching for the older version of the lib.
3) Create a symbolic link to signals lib (for some reason it wasn’t there for me):
   cd /usr/lib/arm-linux-gnueabihf
   ln -s libboost_signals.so libboost_signals.so.1.54.0

You should be ready to build now. You might have to adjust the CMake file, but generally you can use the one I have provided in my repo (https://github.com/morgan1189/skoltech-swarmfleye/tree/master/kinect-webot-on-the-robot/).

In case you don’t yet know how to build your project, do this:
cd ~/<project_folder>
mkdir build
cd build
cmake ..
make

To run just do this:
./<project_name>