Department of Electrical and Electronic Engineering

ELEN90066 Embedded System Design

# Workshop Four (W04)

## Kobuki Navigation in C

Welcome to Workshop 4 for Embedded System Design. In this workshop you will be introduced to the Kobuki robot platform. You will program a state machine that reads sensors and drives a robot with a basic sense of orientation while avoiding obstacles. The programming will be done in C using Microsoft Visual Studio, simulated on the computer using CyberSim and then compiled and deployed to the myRIO using Eclipse, building upon the skills you have learned from the previous workshops.

# Pre-workshop background

Carefully read the following sections in the textbook:

- Chapter 3: Discrete Dynamics, Section 3.3 (Finite-State Machines)
- Chapter 10: Input and Output, Section 10.1.3 (Serial Interfaces)

and be comfortable with the concepts covered before beginning this workshop.

Skim through the Kobuki User Guide to understand its content and layout. You should be able to quickly find relevant information within it.

## Pre-workshop questions

1. Review the template project provided for the workshop exercises in

   **C Statechart\kobukiNavigationStateChart.c**.

   The template code implements a state machine; sketch an FSM diagram representing its behaviour.

2. Does the template code use polling or interrupts to read the Kobuki sensors?

   **Hint:** See the file `kobukiSensorPoll.c`.

   Is this the same mechanism used by the Kobuki protocol?

3. Suppose you want the Kobuki to continue executing a single command for 5 seconds. How might you prevent it from shutting down mid-action?

# Workshop Exercises

The following tasks reference files that are distributed on the LMS as a `.zip` file. You need to download and extract these files to a folder with write permissions in order to complete this workshop. All file paths given are with respect to the top-level folder of the downloaded archive file.

You will need the following equipment and software to complete this workshop, in addition to equipment used in prerequisite workshops (if applicable):

- Computer with Windows 10
- National Instruments LabVIEW 2018 myRIO Toolkit
- National Instruments myRIO 1900
- 12V DC power supply
- USB cable
- C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition 2017-2018
- National Instruments LabVIEW Run Time Engine 2018
- Microsoft Visual Studio 2017 (Enterprise)
- Kobuki robot connected to myRIO via DB25 cable interface

In this workshop you will do the following:

- Compile and run the default program in CyberSim **(TASK 1 - 5 marks)**
- Make the simulated robot navigate obstacles in a simulated environment in CyberSim **(TASK 2 - 10 marks)**
- Transfer the navigation code to the real robot and run it so that the robot successfully avoids obstacles **(TASK 3 - 10 marks)**

**There are 25 marks available for successful completion of all tasks in this workshop, worth 1% of your final mark in the subject.**

## Kobuki Navigation Design Requirements

The "B0" button on the Kobuki is referred to as the "Play" button. For CyberSim, a separate "Play" button has been added on top of the Kobuki top view which maps to the "B0" button.

The following are the design requirements for the obstacle course navigation of the robot.

1. **Startup**: When powered on or reset, the robot shall not move until the "Play" button is pressed.
2. **Run**: The robot shall begin movement the first time the "Play" button is pressed and continue running until paused or stopped.
   (a) **Ground Orientation**: The ground orientation of your robot is the direction the front of the robot is pointing when it first runs. The ground orientation does not change after subsequent pausing/resuming; only after a power cycle, reprogram or restart of the robot or its embedded controller.
   (b) **Drive**: Your robot shall maintain ground orientation and drive forward while on level ground and clear of obstacles.
   (c) **Obstacle Avoidance**: Your robot shall avoid obstacles.
      i. **Cliff Avoidance**: Your robot shall not fall off of cliffs or edges.
      ii. **Wheel Hazard Avoidance**: Your robot shall avoid one or more wheels losing contact with the ground. If a wheel loses contact with the ground, your robot shall attempt to recover and move around the incident hazard.
      iii. **Object Avoidance**: Your robot shall avoid objects in its path. It is acceptable for your robot to touch objects as long as it immediately changes course in an attempt to avoid.
      iv. **Avoidance Robustness**: Obstacle avoidance must always be satisfied, even if multiple obstacles are encountered simultaneously or in short succession.
      v. **Reorientaton**: After avoiding an obstacle, your robot shall eventually reorient to ground orientation.
3. **Pause/Resume**: The "Play" button on the top of the robot shall start/resume or pause movement of the robot. At any point the robot is moving, the "Play" button shall cause it to immediately and completely stop. Subsequently pressing the "Play" button shall cause the robot to resume operation.
4. **Performance**: When moving, your robot must satisfy the following performance characteristics:
   (a) **Turnabout**: Your robot shall never rotate in place more than 180°.
   (b) **Chattering**: Your robot shall not move erratically or exhibit chattering.

(c) **Abnormal Termination**: Your robot shall not abnormally terminate execution, with the exception of power or mechanical failure.

(d) **Obstacle Hugging**: Your robot shall not repeatedly encounter ("hug') an obstacle for the purpose of navigation.

(e) **Timeliness**: Your robot shall achieve its goals in a timely manner.

**Hint:** Your robot does not need to follow a specific path or trajectory, or return to a specific path or trajectory following obstacle avoidance – it need only eventually return to and maintain its original ground orientation.

## Task 1: Program CyberSim from Visual Studio

This workshop task serves as a tutorial on programming the CyberSim simulator using C and Microsoft Visual Studio. You will load a template project and verify your application is loaded by the simulator.

**Microsoft Visual Studio** is the primary development platform for Microsoft Windows software, including desktop, server, tablet and mobile. Visual Studio includes support for multiple programming languages including C and C++.

**CyberSim** is software used to simulate cyber-physical systems using models of both cyber (software) and physical systems. CyberSim is built especially for the workshop tasks in this subject and is the product of a collaboration between the University of California, Berkeley and National Instruments. CyberSim is written in LabVIEW using the **LabVIEW Robotics Simulator**.

Discrete simulations of continuous systems are approximations and are subject to numerical error. These simulations are based on ordinary differential equations (or differential algebraic equations), that are computationally expensive to solve, and numerical accuracy is balanced against real-time performance. The scope of any simulator is limited: one simulation tool may focus on kinematics, another on electricity and magnetism, another on fluid mechanics, another on energy, another on quantum and nuclear dynamics, and so on. Some tools are dedicated to modelling the behaviour of computers themselves!

CyberSim has a physical model of the Kobuki robot. Physical models are based on the Open Dynamics Engine rigid body dynamics software that can simulate robots in a virtual environment and render them in 3D. Kobuki sensor packets are constructed from the state of the simulation and passed into a controller that outputs desired wheel speeds. Effects such as control period, quantisation, and noise are built into the simulator and are meant to closely reflect control software running on an embedded controller interfaced with the Kobuki.

LabVIEW Robotics Simulator provides a graphical interface for customising environments, including adding, removing, and moving obstacles, changing the initial position of the robot, adjusting sensor locations, and adding multiple robots to the environment. Robots modelled in CAD software such as SolidWorks can be imported and customised with sensors and actuators. The configuration of the environment, including dimensions and placement of objects and robots, is stored in an XML file. The resulting XML file may be read in by CyberSim and used to simulate a new environment for further testing.

1. **Launch Visual Studio:** From the Windows Start Menu, launch Visual Studio 2017.
2. **Load the Template C Statechart:** The C Statechart source code and build settings are maintained by a Visual Studio Solution. From the top menu bar, select "File→Open Project..." and navigate to the file

`src\C Statechart\Visual Studio Solution\libstatechart.sln`

The solution references all source files needed to compile a statechart Dynamic Linked Library (DLL) for use with CyberSim.

**Note:** You might be asked to convert some files depending on the version of Visual Studio being used. This should be ok.

3. **Review the Statechart source:** The Solution Explorer pane is used to navigate files in a solution. Open the source file **C Statechart\kobukiNavigationStateChart.c**, the only file you need to modify to change the controller behaviour in CyberSim.

   There are three important sections in the code:

   (a) The state names are defined in an enumerated type at the top of the code. This makes adding extra states simple.

   (b) The state transition equations are defined in a section involving a sequence of conditional statements. Guards specify under which conditions the state will change.

   (c) The state actions are defined using a case statement, which depends on the current state that the robot is in. The state actions ONLY define what the outputs are doing in each state.

4. **Compile:** From the top menu bar, select "Build→Build Solution" to compile the solution. The output directory is automatically set to the CyberSim program folder. Ensure there are no errors in the error list, and following a successful build, check that the output library was written to the CyberSim binary folder in the downloaded files:

   `CyberSim\libstatechart.dll`

5. **Launch CyberSim and Simulate:** Open CyberSim from the downloaded workshop files:

   `CyberSim\CyberSim.exe`

   Open the Configuration tab and in the "Robot State Charts" path field, browse to

   `CyberSim\libstatechart.dll`

   Press Start to begin a simulation. The simulation will progress, executing your controller at each simulation step.

6. **Stop the Simulation:** Before you can change and recompile your controller, you must stop the simulation to unload **libstatechart.dll**. Press the Stop button to stop the simulation. You do not need to exit CyberSim.

7. **Launch a Debug Session (optional):** Visual Studio can attach a debugger to CyberSim to allow you to debug your control software. Visual Studio will launch CyberSim automatically, using a command-line argument to point the simulator to the compiled library. Close all other instances of CyberSim before beginning a debug session.

   In order to debug, compiler optimisations need to be disabled. In the Solution Explorer, right click on the solution name and go to "Properties→Configuration Properties→C/C++→Optimiz and choose "Disabled" from the drop-down list and click OK.

   In Visual Studio, from the top menu bar select "Debug→Start Debugging". The solution should build and launch CyberSim. Press Start to begin a simulation. The debugger will attach when the simulation begins. You may use debugging features such as watch windows and breakpoints to step through your code.

   When you are ready to terminate a debugging session, disable all breakpoints and from the Visual Studio top menu bar and select "Debug→Continue" to allow CyberSim to continue executing uninterrupted. Return to CyberSim and press Exit to close CyberSim, detatch the Visual Studio Debugger, and close the debugging session.

8. **Close CyberSim:** Use the Exit button to terminate CyberSim – do not close the window, as this will terminate CyberSim without unloading your statechart library or the ODE simulator.

Congratulations on using Visual Studio to write a controller for CyberSim!

**TASK 1** Show that you have successfully compiled and run the default program in CyberSim.

**Note :** Your demonstrator will assess this task once you have finished it.

**There are 5/25 marks for successful completion of this task.**

## Task 2: Navigation in C (Simulation)

This task guides you in creating a state machine in C using Microsoft Visual Studio. Your state machine will read sensors and drive the Kobuki with a basic sense of orientation while avoiding obstacles in a simulated environment. This task builds on Task 1.

1. **Launch Visual Studio:** Open Microsoft Visual Studio and import the solution **libstatchart.sln**. Clean and build the solution. The compiled library (DLL file) is written to the `CyberSim` folder.

2. **Execute the Template Code:** Launch `CyberSim.exe`. In the Configuration tab, browse for "Robot State Charts" and select **libstatechart.dll**. Press Start to begin a simulation.

3. **Run a Debug Sequence in CyberSim:** In the Configuration tab, select feedback mode "Debug Navigation" and press Check Grade to run through a sequence of tests that may catch common errors or bugs as you build your solution. Feedback is provided in the "Feedback and Grading" tab.

4. **Navigate in Simulation:** The default algorithm in the C file **kobukiNavigationStateChart.c** only drives in a square and does not contain any obstacle avoidance code. You now need to modify this code so that the robot avoids obstacles and maintains a basic sense of orientation. You must satisfy all design requirements as specified in the Kobuki Navigation Design Requirements.

   **Hints:**

   - You only need to edit **KobukiNavigationStateChart.c**
   - You can detect an object to avoid by using the three bump sensors on the front of the robot and responding when an object has been bumped into.
   - The Kobuki sensor data are available via the `sensors` variable, which has the type `const KobukiSensors_t`, a `struct` that stores the sensor data in the format shown in Figure 1. For example, the Kobuki sensor reading for the (pure) left bump sensor is available at

     `sensors.bumps_wheelDrops.bumpLeft`

     See the header file **KobukiSensorType.h** for details.

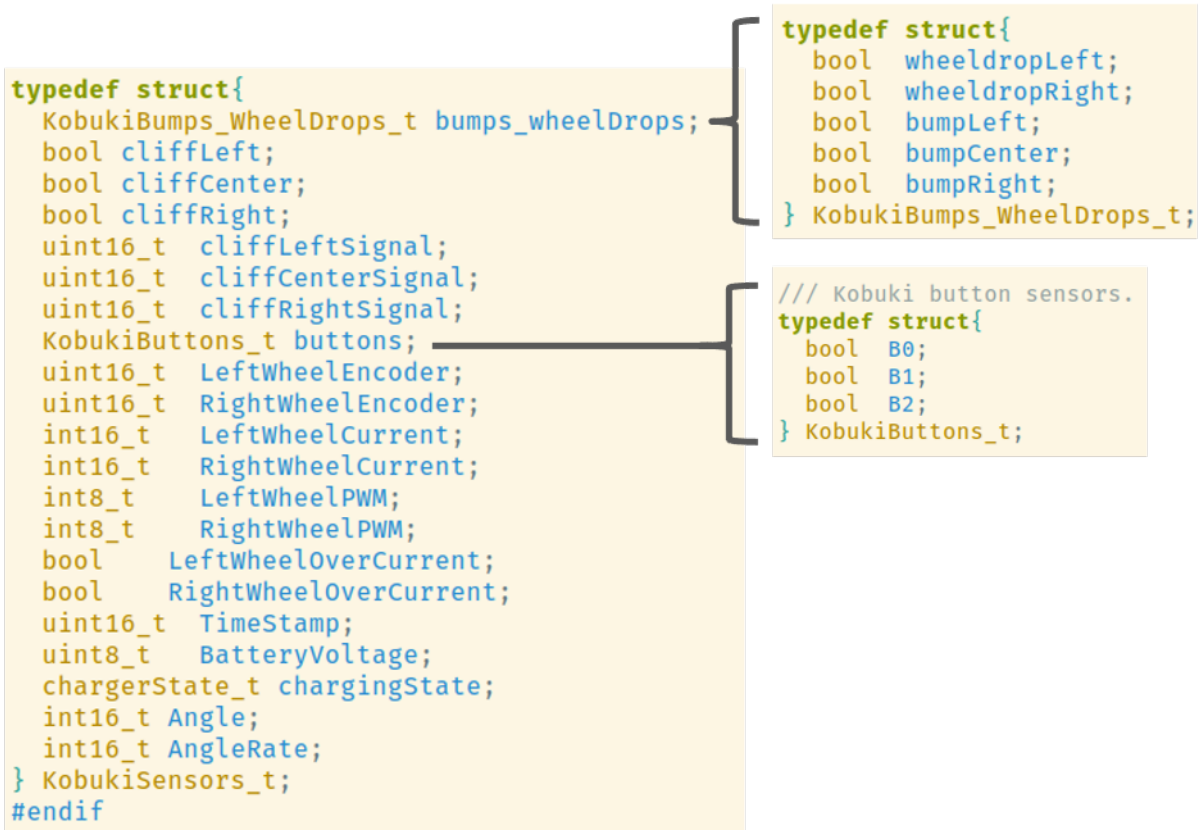


Figure 1: `KobukiSensors_t` struct

- Add your state machine state names to the `enum` list at the top of the C file.
- Add any additional state transition equations in the section containing the sequence of conditional statements.
- Ensure that you define what the outputs, i.e. the wheel motors, are for each state in the case statement block..

**TASK 2** Show that the simulated robot successfully navigates obstacles in a simulated environment in CyberSim using your modified C code.

You must be able to answer the following questions when demonstrating your robot in CyberSim to your demonstrator:

1. Describe (and sketch) your obstacle avoidance algorithm.
2. Did you follow a state machine architecture? If so, which states did you add to the default project?

**Note :** Your demonstrator will assess this task once you have finished it.

**There are 10/25 marks for successful completion of this task.**

## Task 3: Navigation in C (Deployment)

After testing your code in simulation (Task 2), you will now be deploying your code to the Kobuki and testing it in the real world. Your state machine will read sensors and drive the Kobuki with a basic sense of orientation while avoiding obstacles.

1. **Setup myRIO:** Ensure myRIO is powered and connected to your computer.
2. **Verify the Robot Setup:** Your Kobuki is interfaced with a myRIO via a DB25 connector for data signals and a power connector to provide power from the Kobuki to the myRIO. Ensure that these connections are in place.
3. **Launch Eclipse:** Open C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition 2017-2018 and open your existing workspace from previous workshops. Import the project archive **src\kobukiNavigation.zip** by going to "File→import
4. **Set compiler options:** You need to need to replicate the project compiler settings from Workshop 2 by right clicking on the project and choosing "Properties". A reminder of the settings to check:
   (a) "C/C++ Build → Discovery Options", Compiler invocation command.
   (b) "C/C++ Build → Settings", Tool Settings tab, "Cross Settings" - Prefix and Path.
   (c) "C/C++ Build → Settings", Tool Settings tab, "Cross GCC Compiler → Miscellaneous", other flags.
   (d) "C/C++ Build → Tool Settings tab, "Cross GCC Linker → Miscellaneous", linker flags.

   Once you have saved these for the project in the workspace, you will not need to change them again.
5. **Migrate your C code:** Paste your CyberSim/Visual Studio code into **KobukiNavigationStateChart.c**, replacing everything in that file.

   You need to make some small changes to some data types in order to make your code compile in Eclipse for the myRIO:

   (a) `int16_t` to `short`
   (b) `int32_t` to `int`
   (c) `int64_t` to `long int`

   This can be easily done by pasting the following code at the start of the C source file:

```
#define int16_t short
#define int32_t int
#define int64_t long int
```

6. **Build the project in Eclipse:** Double-check that the `MyRio_1900` symbol is set (see Workshop 2) and that the target has been correctly configured. Clean and build all projects and check for any errors before proceeding.

7. **Create a target connection:**, Connect to the remote target by opening the Remote System Explorer pane, or if a connection does not exist, by creating a new remote System Connection (see Workshop 2).

8. **Upload new FPGA fixed personality to myRIO:** You now need to ensure a new FPGA Fixed Personality is on the myRIO. In the Remote System Explorer pane, expand "myRIO→Sftp Files→Root" to see the filesystem on myRIO. Navigate to the folder

   `/var/local/natinst/bitfiles`

   or create the `bitfiles` folder if it does not exist. Delete all `.lvbitx` files in this folder.

   Right-click on the "bitfiles" folder and select "Export from Project...". Navigate to the source folder for the Kobuki Navigation project, go to the `myrio` sub-folder and check to include

   `NiFpga_myRio1900Fpga20.lvbitx`

   No other files or folders should be checked.

   Verify that the destination folder indicates the "bitfiles" folder. Click Finish to transfer the file, and verify the file now appears in the Remote Systems pane under the "bitfiles" folder.

9. **Load the program onto myRIO:** This time, rather than using a Debug/Run Configuration as in a previous workshop, we are going to use a different method to download and run the code on myRIO.

   Firstly, now that we have a binary file compiled, let's use SSH File Transfer Protocol (SFTP) to put it on the Kobuki:

   (a) Ensure the myRIO is connected in the Remote System Explorer.

   (b) Expand "myRIO→Sftp Files→My Home" and create a new folder under "My Home" by right-clicking, called `W04`.

   (c) In the Project Explorer pane, navigate to the binary file under "kobukiNavigation → Binaries". It should be titled `kobukiNavigation - [arm/le]`. Right click and copy the file.

(d) Back in the Remote System Explorer, navigate back to the `W04` folder and right click to paste the binary file into it.

10. **Execute the program directly on the myRIO:**

(a) Launch a new Terminal in the Remote System Explorer.

(b) In the terminal, change folder to `W04` by entering the command `cd W04`.

(c) Change the binary file's permissions to be executable by entering:

```
chmod +x kobukiNavigation
```

(d) Run the executable file as background process:

```
./kobukiNavigation &
```

Take note of the 4-digit number that is returned when you run the executable. This is its *process ID*.

(e) Disconnect the myRIO from the USB cable and place the Kobuki on the floor near an obstacle.

(f) Press Button "B0" to pause / run your code on the Kobuki.

(g) If the myRIO loses power you will need to reconnect via Eclipse and restart the executable.

***Hint:*** *When revising your code and downloading a new binary to the myRIO, you need to ensure that the existing program has terminated. You can check if the process is still running by typing the command* `ps` *in the terminal when connected to the myRIO and looking for the executable name.*

If you find the executable name in the list of processes, you can kill the process by typing `kill`— followed by the process ID, e.g.

```
kill 2198
```

**TASK 3** Refine your C code to program the real robot to avoid obstacles and maintain a basic sense of orientation. You must satisfy all design requirements as specified in the Kobuki Navigation Design Requirements.

**Note :** Your demonstrator will assess this task once you have finished it.

**There are 10/25 marks for successful completion of this task.**