ELEN90066 Embedded System Design

Workshop 4 – Kobuki Navigation in C

Workshop 4 – Kobuki Navigation in C

The goal is to design a control algorithm for the Kobuki robot according to *model-based design* principles. After a successful design and simulation, we will deploy our code directly on the myRIO board which will control the Kobuki.

NOTE: These slides are only a SUPPLEMENT to the main workshop notes on LMS. Please follow the instructions in there!

Workshop 4 marking

- There are three tasks to complete in the workshop today, total of 25 marks
  - TASK 1 : Running the default program in CyberSim: 5 marks
  - TASK 2 : Making the simulated robot do something other than the default behaviour in CyberSim 10 marks
  - TASK 3 : Transferring the navigation code to the real robot and running it so that the robot successfully avoids obstacles. 10 marks

    Definition of successful obstacle avoidance: When the robots hits an obstacle, it goes around and continues in a line parallel with its original path of motion.

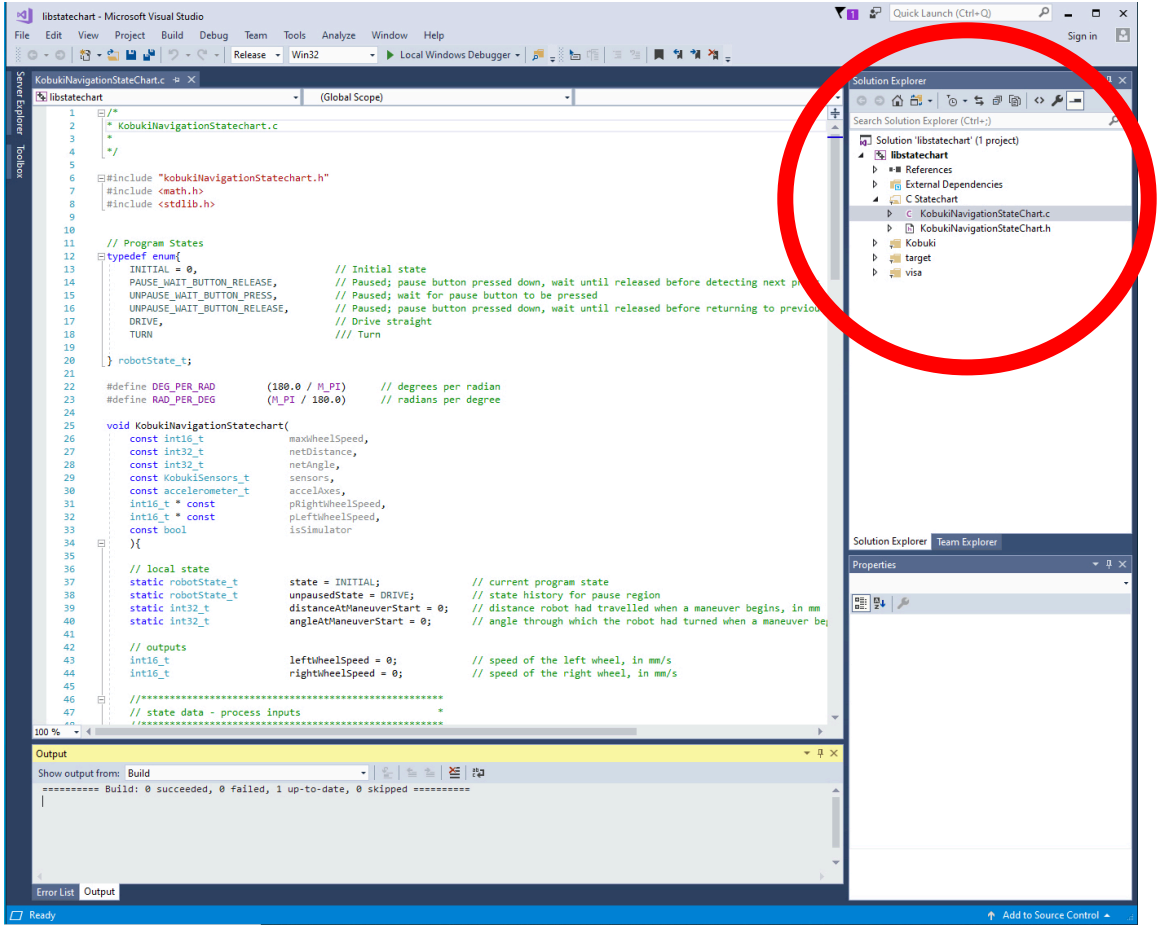
Please notify me every time you finish one of the assessed tasks, so I can check that off for you!

Task 1 – Program CyberSim

3 : Review the Statechart source

The Solution Explorer pane is used to navigate files in a solution.

Open the source file `kobukiNavigationStateChart.c`, the only file you need to modify to change the controller behaviour in CyberSim.



Task 1 – Program CyberSim

3 : Review the Statechart source

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

```
// Program States
typedef enum{
    INITIAL = 0,                        // Initial state
    PAUSE_WAIT_BUTTON_RELEASE,          // Paused; pause button pressed down, wait until released before detecting next press
    UNPAUSE_WAIT_BUTTON_PRESS,          // Paused; wait for pause button to be pressed
    UNPAUSE_WAIT_BUTTON_RELEASE,        // Paused; pause button pressed down, wait until released before returning to previous
    DRIVE,                              // Drive straight
    TURN                                /// Turn
```

The state transition equations are defined in this section. Guards specify under which conditions the state will change.

```
//************************************
// state transition - run region      *
//************************************
else if (state == DRIVE && abs(netDistance - distanceAtManeuverStart) >= 250){
    angleAtManeuverStart = netAngle;
    distanceAtManeuverStart = netDistance;
    state = TURN;
}
else if (state == TURN && abs(netAngle - angleAtManeuverStart) >= 90){
    angleAtManeuverStart = netAngle;
    distanceAtManeuverStart = netDistance;
    state = DRIVE;
}
// else, no transitions are taken
```

Task 1 – Program CyberSim

3 : Review the Statechart source

The state actions are defined using a case statement, which depends on the current state the robot is in.

The state actions ONLY define what the outputs are doing in each state.

Outside the case statement, the wheel speeds are set by setting a pointer value equal to the variable set in the case statement.

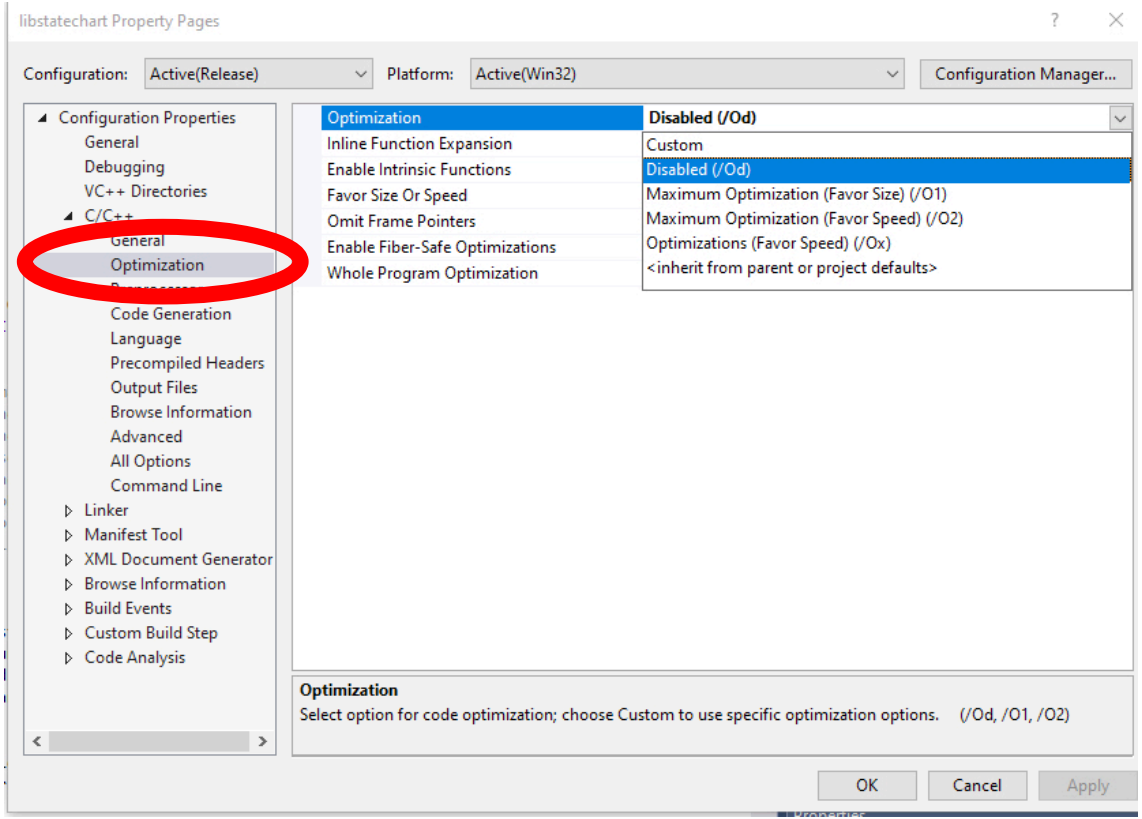
```
//*****************
//* state actions *
//*****************
switch (state){
case INITIAL:
case PAUSE_WAIT_BUTTON_RELEASE:
case UNPAUSE_WAIT_BUTTON_PRESS:
case UNPAUSE_WAIT_BUTTON_RELEASE:
    // in pause mode, robot should be stopped
    leftWheelSpeed = rightWheelSpeed = 0;
    break;

case DRIVE:
    // full speed ahead!
    leftWheelSpeed = rightWheelSpeed = 100;
    break;

case TURN:
    leftWheelSpeed = 100;
    rightWheelSpeed = -leftWheelSpeed;
    break;

default:
    // Unknown state
    leftWheelSpeed = rightWheelSpeed = 0;
    break;
}


*pLeftWheelSpeed = leftWheelSpeed;
*pRightWheelSpeed = rightWheelSpeed;
```

Task 1 – Program CyberSim

7 : Launch 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 → Optimization’” and choose “Disabled" from the drop-down list and click OK.



Task 2 – Navigation in C (Simulation)

4 : Navigate in Simulation

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 the figure.

For example, the Kobuki sensor reading for the (pure) left bump sensor is available at:

`sensors.bumps_wheelDrops.bumpLeft`

```
typedef struct{
  KobukiBumps_WheelDrops_t bumps_wheelDrops;
  bool cliffLeft;
  bool cliffCenter;
  bool cliffRight;
  uint16_t  cliffLeftSignal;
  uint16_t  cliffCenterSignal;
  uint16_t  cliffRightSignal;
  KobukiButtons_t buttons;
  uint16_t  LeftWheelEncoder;
  uint16_t  RightWheelEncoder;
  int16_t   LeftWheelCurrent;
  int16_t   RightWheelCurrent;
  int8_t    LeftWheelPWM;
  int8_t    RightWheelPWM;
  bool    LeftWheelOverCurrent;
  bool    RightWheelOverCurrent;
  uint16_t  TimeStamp;
  uint8_t   BatteryVoltage;
  chargerState_t chargingState;
  int16_t Angle;
  int16_t AngleRate;
} KobukiSensors_t;
#endif
```

```
typedef struct{
  bool  wheeldropLeft;
  bool  wheeldropRight;
  bool  bumpLeft;
  bool  bumpCenter;
  bool  bumpRight;
} KobukiBumps_WheelDrops_t;
```

```
/// Kobuki button sensors.
typedef struct{
  bool  B0;
  bool  B1;
  bool  B2;
} KobukiButtons_t;
```

Task 2 – Navigation in C (Simulation)

4 : Navigate in Simulation

The header file **KobukiSensorType.h** gives details on the sensors, their variable names, types, and values returned.

```
/// accelerometer values
typedef struct{
    double  x;                          ///< x axis, in g
    double  y;                          ///< y axis, in g
    double  z;                          ///< z axis, in g
} accelerometer_t;

typedef struct{
    bool    wheeldropLeft;      ///< State of the left wheeldrop sensor; when true, indicates the wheel is not in contact with the ground.
    bool    wheeldropRight;     ///< State of the right wheeldrop sensor; when true, indicates the wheel is not in contact with the ground.
    bool    bumpLeft;           ///< State of the left bump sensor; if true, the left side of the robot is in contact with an object.
    bool    bumpCenter;         ///< State of the right bump sensor; if true, the right side of the robot is in contact with an object.
    bool    bumpRight;          ///< State of the right bump sensor; if true, the right side of the robot is in contact with an object.
} KobukiBumps_WheelDrops_t;
/// Kobuki button sensors.
typedef struct{
    bool    B0;
    bool    B1;
    bool    B2;
} KobukiButtons_t;



typedef enum { DISCHARGING, DOCKING_CHARGED, DOCKING_CHARGING, ADAPTER_CHARGED, ADAPTER_CHARGING } chargerState_t;
typedef struct{
    ///  bump and wheel-drop sensors.
    KobukiBumps_WheelDrops_t bumps_wheelDrops;
    /// Binary interpretation of the infrared wall sensor; if true, the sensor has read above a predefined threshold and indicates a wall (or
    /// Binary interpretation of the left cliff sensor; if true, the sensor has read below a predefined threshold and indicates the sensor is
    bool cliffLeft;
    /// Binary interpretation of the front left cliff sensor; if true, the sensor has read below a predefined threshold and indicates the sens
    bool cliffCenter;
    /// Binary interpretation of the right cliff sensor; if true, the sensor has read below a predefined threshold and indicates the sensor is
    bool cliffRight;
    uint16_t    cliffLeftSignal;        ///< Sampled value of the analog left cliff sensor, range 0-4095.
    uint16_t    cliffCenterSignal;  ///< Sampled value of the analog front left cliff sensor, range 0-4095.
    uint16_t    cliffRightSignal;       ///< Sampled value of the analog right cliff sensor, range 0-4095.
```

Task 3 – Navigation in C (Deployment)

8 : Upload new FPGA Personality

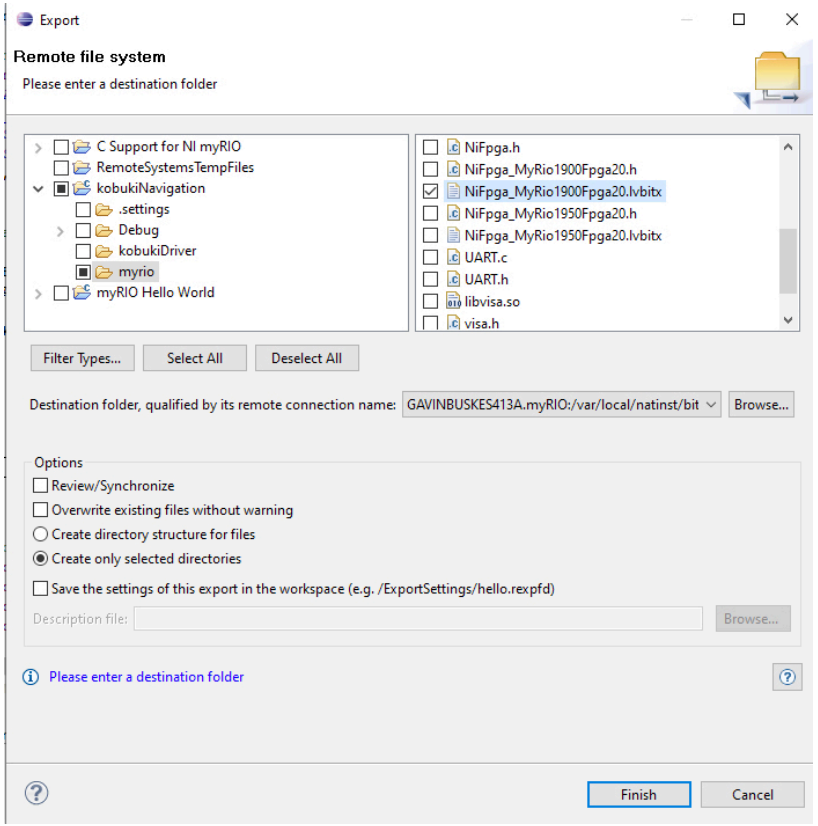
You now need to ensure a new FPGA Fixed Personality is on the myRIO.

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.



Task 3 – Navigation in C (Deployment)

9(c) : Load the Program

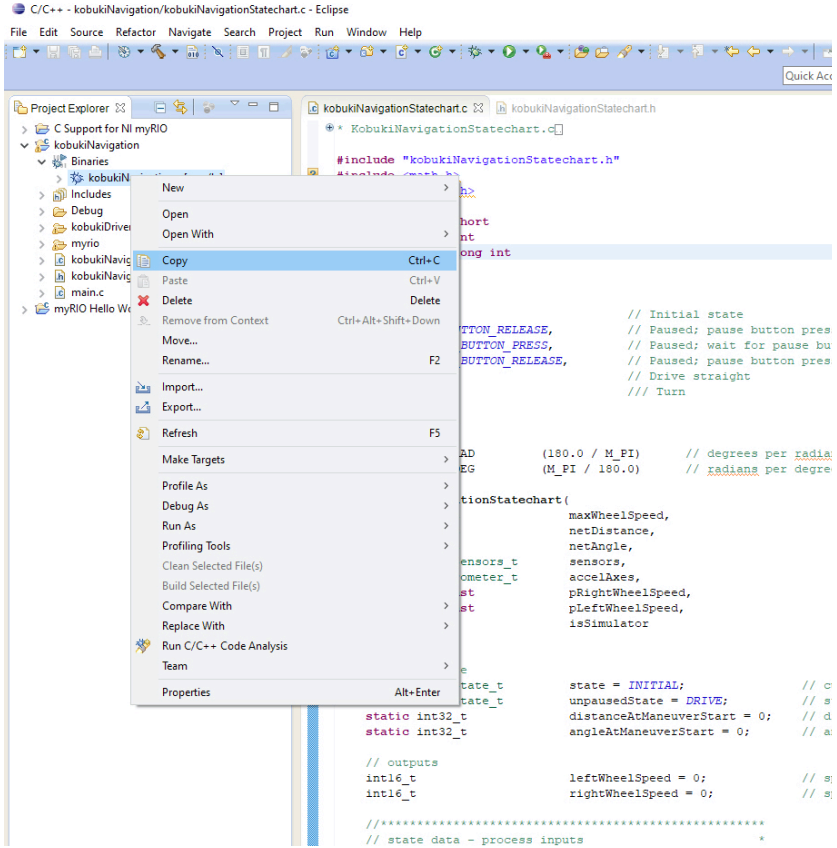
In the Project Explorer pane, navigate to the binary file under

“kobukiNavigation → Binaries”

The binary should be titled

kobukiNavigation - [arm/le]

Right click and copy the file.



Task 3 – Navigation in C (Deployment)

10(d) : Execute the program

Run the executable file as background process:

```
admin@NI-myRIO-1900-0319ad83:~/W04# ./kobukiNavigation &
[1] 2198
```

./kobukiNavigation &

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