Embedded Systems Engineering

Software Engineering Lab 1

Jos Onokiewicz

4 April 2018
version 0.8

Document history

Modified	Version	Modified by	Updates
			…
31-01-2018	0.6	Jos Onokiewicz	Updated the C++ assignments. Added some initial versions for the UML, Unit test and IoT MQTT assignments.
07-03-2018	0.7	Jos Onokiewicz	Solved some typos. Some assignments extended. Added some more MQTT assignments.
04-04-2018	0.8	Jos Onokiewicz	Removed Unit testing assignments. Assignment 11 is now optional. Added more development code details to the texts of assignments 4 and 5.

Hogeschool van Arnhem en Nijmegen

Embedded Systems Engineering

INHOUD

PREFACE

This document contains the practical assignments that belong to the first block of SE. We assume sufficient experience in the use of QtCreator and basic level C++ programming using classes and objects. The use of Modern C++ (C++11 and C++14) is a starting point for the assignments. The Roomba vacuum cleaner robot is used as an existing target for the reverse-engineering assignment. An introduction to IoT using the MQTT protocol (based on the Mosquitto library) is added to the assignments.

Arnhem, Januari 2018
Jos Onokiewicz

1. Preparation of the development environment

This lab will focus on Modern C++ programming and system development using UML.

1.1 The transition to Modern C++

C++ have been improved several times. New standards became available in 2011, 2014 and 2017: C++11, C++14 and C++17. These three are often referred to as Modern C++.

Bjarne Stroustrup, the creator of C++, said that C++11 "Feels like a new language, the pieces just fit together better." Compiler support for C++11: GCC 4.8 and later completely supports it. We mainly focus us on C++11 and some C++14. Full compiler support for C++14: GCC 5.0 and later.

Because there are several C++ standards, you must choose a compiler able to compile according to a chosen C++ standard.

If you become fluent with C++ you need only a few weeks to become proficient in for example Python, Java or C#.

1.2 Necessary tooling

The next tools must be installed under Linux:

- Qt Creator (version >= 4.4.1 using Qt version >= 5.9.2)
- Doxygen and doxywizard (for generating code documentation in html-format)
- plantUML plugin for the Chrome webbrowser
- Git
- Valgrind
- MQTT Mosquitto library

1.3 Drawing UML diagrams using plantUML

Drawing UML diagrams without much lay-out effort is very useful, especially in an agile project management approach.

It will take some time to learn the plantUML scripting language for drawing diagrams. For that reason several script for different diagrams examples are available. These examples can be easily updated to your own needs. A plantUML plugin is available for the Chrome web browser.

Examples plantUML scripts and generated diagrams: http://plantuml.com/

1.4 Creating code documentation using Doxygen

"Doxygen is the de facto standard tool for generating documentation from annotated C++ sources, but it also supports other popular programming languages such as C, Objective-C, C#, PHP, Java, Python, IDL (Corba, Microsoft, and UNO/OpenOffice flavors), Fortran,

VHDL, Tcl, and to some extent D.

Doxygen can help you in three ways:

1. It can generate an on-line documentation browser (in HTML) and/or an off-line reference manual LaTeX from a set of documented source files. There is also support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages. The documentation is extracted directly from the sources, which makes it much easier to keep the documentation consistent with the source code.

2. You can configure doxygen to extract the code structure from undocumented source files. This is very useful to quickly find your way in large source distributions. Doxygen can also visualize the relations between the various elements by means of include dependency graphs, inheritance diagrams, and collaboration diagrams, which are all generated automatically.

3. You can also use doxygen for creating normal documentation (as I did for the doxygen user manual and web-site).

Doxygen is developed under Mac OS X and Linux, but is set-up to be highly portable. As a result, it runs on most other Unix flavours as well. Furthermore, executables for Windows are available."

Source: http://www.stack.nl/~dimitri/doxygen/

2. C++ assignments

This chapter contains the practical assignments to be made. Please note that we often have to use the programming knowledge gained in a previous assignment for each subsequent assignment. Consult the mentioned topics in prescribed books, YouTube video's and code examples as indicated in #OnderwijsOnline.

2.1 Preparation of the assignments

You are expected to prepare the assignments outside of the scheduled lab hours as much as possible in order to be able to use the available lab time as efficiently as possible. You carry out the assignments on your own laptop on which QtCreator, g++ and plantUML is installed under Linux. Collaboration with other students is explicitly recommended.

2.2 C++ and Doxygen assignments

Use C++ and avoid the usage of C library functions. Answer the questions in brief comments in the code, for example below the body of the main() function.

The assignment texts contain references to chapters or paragraphs in the C++ book 'C++ Primer Plus' by Stephen Prata. This book should be considered as a C++ reference manual besides all other selected internet resources. Train yourself in searching in these sources for finding answers to your programming questions. The C++ book does not yet cover all the new C++11 language features. For this reason, we must also consult other C++ references.

!	Use the programming style and naming conventions as used in the provided code examples. The code examples in the C++ book differ in some programming style details with the provided code examples. Do not use standard C-functions anymore. Clean Code: write clean, maintainable and robust code. Always care about your code quality! Writing clean code always pays off. Video (56:22): JAMES MCNELLIS & KATE GREGORY "MAKING C++ CODE BEAUTIFUL"

!	Always distribute your code over multiple files. Every class should be coded in a .h and .cpp file. Use the name of the class for the names of the .h and .cpp file. In QtCreator you can create these files by choosing the Class option.

The next following programming assignments are somewhat simplified to be able to make them in a short time. In real development projects, the created classes should contain more member functions and error handling.

C++ Assignment 1. Get acquainted with some STL sequence container classes.

The Standard Template Library (STL) contains several sequence container classes. These classes are implemented in the std namespace. Focus on inserting, removing and changing the data in the containers. Avoid the literal copying of code from mentioned sources.

a. Instantiate two `std::string` objects. Initialise these strings with some text. Use several member functions and operators to insert, remove and change the contents of the strings. Also, copy and compare two strings. Show the results of the invoked member functions and operators by using the cout output stream.

b. Instantiate two `std::vector` objects containing int's. Do the same as in a. Show the contents of every vector by using a range-based for-loop, see[C++, p. 991-992], and the cout output stream.

c. Instantiate two `std::list` objects containing double's. Do the same as in a. Also determine if two lists contain the same data contents.

d. Instantiate several `std::vector` of `std::strings`. Extend the next code example with more member function calls and operator usage.

```
// Using the default constructor.
vector<string> vdata1;
// Using an initialiser list for initialisation.
vector<string> vdata2{"start", "setspeed 2", "stop"};

vdata1 = vdata2;

cout << vdata1.size() << endl;
cout << vdata1[1] << endl;
```

e. Show how the `std::queue` class (container adapter, a FIFO data structure) can be used.

C++ Assignment 2. Command handling for controlling a rotation motor

An embedded system must be controlled by text commands. The application must be able to read in `main()` an input line containing a command(use `cin` as input stream). After reading a command the code must decide which command has been entered and the related function should be executed.

The next 5 unique commands for controlling the rotation of a motor:

- `start`
- `shutdown`
- `restart`
- `incRPM` `increment RPM by some constant value`
- `decRPM` `decrement RPM by some constant value`
- `getRPM`

Only after entering the command `start` the other commands will be enabled to execute. If `start` is not used already or an unknown command is entered an appropriate error message must be showed in the text console to the user. Use the output stream *cerr* for error messages.

a. Implement the class RotationMotor with an appropriate constructor, a data member representing the RPM and member functions related to the mentioned commands. These member functions should show a text containing the name of the command and the value of RPM if executed.

b. Add a getter function for RPM. In what way should you use `const`?

c. The command `restart` will set RPM to 0. The command `shutdown` will set RPM to 0 and the program will be ended.

d. The class RotationMotor must contain three constants for the maximum positive and negative value of RPM, and the value for incrementing and decrementing RPM. If `incRPM` and `decRPM` will result in an RPM value parameter outside this range, RPM must be set to the closest value of these maximum positive or negative constants. Implement these constant data members as `static const`, see [C++, p.628-632].

   Naming convention constants: all upper case.

C++ Assignment 3. Use the class array, auto declaration and a range based for loop

Implement the next code block in a C++ project in `main()` using the STL array template class (container class).

The constructor for `ar` uses list-initialisation, see [C++, p. 187-188, 355-356].

For traversing the array a range-based for loop and an `auto`-declaration is used, see [C++, p. 109].

- What is the output of the next code block?

```
using namespace std;
// C++11 STL array template class, parameters between < and >
// Using C++11 uniform initialisation by an initializer_list {…}
array<int, 3> ar{2, -2, 3};

// Range-based for loop, el is not the index of a specific
// element in the array!
// e1 is a reference to an array element in ar (we used &),
// now we can change the contents of the array.
// auto declaration: type will be deduced by the compiler.
for (auto& el: ar)
{
   cout << el << " ";
   el *= 10;
}
cout << endl;

ar[0] = 10 * ar[1];

for (const auto el: ar)
{
   cout << el << " ";
}
```

- What is the output of the code code block if we remove `&` in the second line? Why?
- Why should you use `const` in the second for loop?
- The second range-based for-loop does not contain an `&`. What will happen now for `el`?
- We can also use `ar.at(1)` instead of `ar[1]`. What is the difference? See for a string example using `at()`, see [C++, p.1259].

a. Show in the code that you can copy `ar` to a new array of the same type using an assignment. Could you do the same with a classic C-array?

b. Create an array of 4 RotationMotor objects (see assignment 2, add the .h and .cpp files to your C++ project).

c. Use a range based for loop for setting the RPM of all motors to the same chosen value. Avoid the copying of the RotationMotor objects in the range-based for-loop by using `&`.

C++ Assignment 4. Implement the Dataframe class for Roomba communication

The Roomba (robotic vacuum cleaner) can be connected by a serial link. Commands and system state information can be serially communicated by a UART using a data frame containing bytes. A returned data frame should follow some rules, using: a known header byte, a byte representing the length of the data frame, some data and a checksum byte. The checksum is necessary to check if a received data frame is not corrupted. In this assignment we are going to implement the class Dataframe for managing the data in a data frame.

Use the C++ STL container class `std::array<std::uint8_t, 255>` for implementing such a data frame as a data member in the class Dataframe (composition).

- This data frame has a fixed size of 255 bytes.
- Use for these bytes a fixed width integer type `uint8_t` from C++11.

Use the new C++11 `typedef` notation that is generally much easier to understand:

```
using dataframe = std::array<uint8_t, 255>;
```

This will increase the code quality by improving the code readability. Put this line of code in the public part of the class Dataframe. Because `dataframe` is in the scope of the class, you must use: `Dataframe::dataframe`.

The iRobot® Roomba Open Interface (OI) Specification [page 18] shows an example for calculating the checksum of a stream of data packets returned after requesting sensor data by the next command bytes (input command): `[148][2][29][13]`

All values are showed in the decimal format.

The format of the data returned is:

```
[19][N-bytes][Packet ID 1][Packet 1 data…][Packet ID 2][Packet 2 data…][Checksum]
```

N-bytes is the number of bytes between the N-bytes byte and the checksum.

The checksum is a 1-byte value. It is the 8-bit 2-complement of all of the bytes between the header and the checksum. That is, if you add all of the bytes after the checksum, and the checksum, the low byte of the result will be 0.

Example:
To get data from Roomba's left cliff signal (packet 29) and virtual wall sensor (packet 13), send the following command string to Roomba:

```
[148] [2] [29] [13]
```

NOTE:
The left cliff signal is a 2-byte packet and the virtual wall is a 1-byte packet. Roomba starts streaming data that looks like this:

```
[19] [5] [29] [2] [25] [13] [0] [182]
```

header N-bytes packet ID 1 Packet data 1 (2 bytes) packet ID 2 packet data 2 (1 byte) Checksum

NOTE:
Checksum computation:

```
(5 + 29 + 2 + 25 + 13 + 0 + 182) = 256 and (256 & 0xFF) = 0
```

a. The default constructor of the class Dataframe must initialise all data frame elements to 0.

b. The next class member function must initialise all data members of the data frame. This input array should contain only data without the header data, the size and the checksum:

```
void Dataframe::setData(const Dataframe::dataframe &data,
                        std:uint_8 size)
```

The array `data` can contain:

```
[29] [2] [25] [13] [0]
```

The input parameter `size` is necessary to identify how many bytes in `data` are relevant. The input parameter `size` equals 5 in this example. This function should add the appropriate header data, size and checksum to the contained data frame:

```
[19] [5] [29] [2] [25] [13] [0] [182]
```

c. Implement the next class member function for calculating the checksum and storing the checksum value in the contained data frame:

```
void Dataframe::setChecksum();
```

d. Implement a setter function for the contained data frame. The input parameter is dataframe typed. This data frame should already contain the header data, size and checksum data. This could be a received data frame whose checksum still needs to be tested.

```
void Dataframe::setDataframe(
    const Dataframe::dataframe &dataframe);
```

e. Implement a getter function for the contained data frame.

f. Implement the next member function for showing the contents of the data frame:

```
void Dataframe::show()const;
```

Format:

```
[19] [5] [29] [2] [25] [13] [0] [182]
```

Because the data is typed as `uint8_t` you must use `static_cast<int>()` to print the data in decimal format. Use the size value in the second byte in the data frame to print only the data up to and including the checksum value.

g. Implement the next member function for showing the all members (255) of the contained data frame:

```
void Dataframe::showAll()const;
```

h. Add the next class member function for determining if the checksum value in the contained data frame is correct:

```
bool Dataframe::checksumIsCorrect() const;
```

i. Implement the next member function for clearing the data frame (all 255 data members should become 0):

```
void Dataframe::clear();
```

j. Create a Dataframe object in `main()` and use all implemented public class member functions to show the correctness of your code.

Doxygen: put above the class code some Doxygen formatted comments describing the responsibilities of the class Dataframe and the member functions. Create an additional directory besides the C++ QtCreator project directory for storing the generated HTML documentation and the Doxyfile containing the Doxygen configuration data.

Example: project directory is named Dataframe, the Doxygen directory should be named Dataframe-docs.

C++ Assignment 5. Implement the CommandProcessor class

This assignment is an extension of C++ Assignment 2. Copy this code into a new QtCreator C++ project.

In C++ no switch-case statement is available for strings because a switch-case statement can only be used for integral types. Therefore we need to use a lot of nested if-else statements for the comparison of strings and executing the related functions. We can avoid this approach by using the STL `std::map` class.

Code example: AdvancedCPP/pairs-maps

Because we can only use one type for the function pointers in the map, we must wrap some functions in a function according to the type used in the map.

a. Implement the class CommandProcessor containing an STL `std::map` data member `commands_` for mapping command names (strings) to command function. Put the next line in the public part of the class:

```
using commandfunction = std::function<void()>;
```

Put the next line in the private part of the class:

```
std::map<std::string, commandfunction> commands_;
```

b. The command function `getRPM` must be wrapped in a function using the signature (interface) `std::function<void()>`. This function will call the `getRPM` function and write the returned result to cout.

c. Implement a default constructor, initialising an empty std::map.

d. Implement the next class member function for adding new pairs of command strings and related functions:

```
void CommandProcessor::
     addCommand(const std::string &command,
                CommandProcessor::commandfunction cf);
```

e. Implement the next class member function:

```
void CommandProcessor::
     executeCommand(const std::string &command)
{
   // p is an iterator
   auto p = commands_.find(command);
   if (p != end(commands_))
   {
      auto commandfunc = (*p).second;
      commandfunc();
   }
   else
   {
      …
   }
```

```
    }
```

By using inside this function the `find()` member function of `std::map` we get an iterator to the corresponding pair in the map. By dereferencing this iterator we can execute the function by using the second member of this pair. If not found we get an end pointer value. In this case we cannot execute a command. An error message should be showed.

f. Instantiate in main() a RotationMotor and CommandProccesor object.

Example of adding a pair of a command string and related function to the map in the CommandProcessor object:

```
int main()
{
   RotationMotor rotMotor;
   CommandProcessor cmdp;

   cmdp.addCommand("start",
                   bind(&RotationMotor::start, &rotMotor));

   …
```

Implement the input of string commands and the related executing of RotationMotor functions. Test your code for all commands.

Doxygen: put above the class code some Doxygen formatted short comments describing the responsibilities of the class. Use the same approach as in assignment 4.

C++ Assignment 6. Implement the CartVec2D class representing a 2D Cartesian vector

In Cyber-Physical Systems (CPS) "engineered systems that are built from, and depend upon, the seamless integration of computational algorithms and physical components" (National Science Foundation) for example robots, autonomous vehicles and automatic pilots, vector math plays an important role for describing the measured and controlled position, speed and acceleration for this kind of systems.

See for CPS:

- http://cyberphysicalsystems.org
- https://www.nist.gov/el/cyber-physical-systems
- A free course, online lecture notes: https://bit.ly/LNCPS-2014
- Example like a Semester4 project: https://www.youtube.com/watch?v=U7y1wiYqHDc

"An embedded system is a computational system embedded in a physical system. Any Cyber-Physical System contains an embedded system. The main distinction is that the term “embedded system” reflects a primary focus on the computational component (that is embedded in a larger, physical system). The CPS view emphasizes the importance of taking into account the physical context of the computational system which is often necessary to design, test, and verify the functionality that we are developing."

A Cartesian vector in 2D contains the x and y coordinate. We can do some arithmetic with vectors. For example adding or subtracting two vectors to each other.

See for vector math:

- https://www.mathsisfun.com/algebra/vectors.html

A binary operator uses 2 operands. For example the + and * operator:

```
result = lhs + rhs;

result = lhs * rhs;
```

- **rhs**: right-hand sight operand of the binary operator
- **lhs:** left-hand sight operand of the binary operator

The next code example shows the CartVec2D class partly implemented in the header file. Coding style: put all friend functions in the class above the public part. Friend functions are declared inside the class but they are not class members.

```
/// CartVec2d represents a 2D Cartesian vector.
class CartVec2D
{
   friend std::ostream& operator<<(std::ostream &os, const CartVec2D &rhs);
   friend CartVec2D operator+(const CartVec2D &lhs, const CartVec2D &rhs);
   friend CartVec2D operator-(const CartVec2D &lhs, const CartVec2D &rhs);

public:
   CartVec2D(double x, double y): ……
   /// Using C++11 ctor delegation for the default ctor.
   CartVec2D(): CartVec2D(0.0, 0.0){}
   /// Default copy ctor.
```

```
   CartVec2D(const CartVec2D &other) = default;
   /// Default assignment operator.
   CartVect2D& operator=(const CartVec2D &other) = default;
   /// Default dtor.
   ~CartVec2D() = default;

   // += and -= operators are class member functions.
   CartVec2D &operator+=(const CartVec2D &rhs);
   CartVec2D &operator-=(const CartVec2D &rhs);

   /// Magnitude of vector.
   double length() const;

   ……

private:
   double x_;  // Naming convention class data members, use _ suffix.
   double y_;
};
```

Coding style: put the implementation of all functions in the CartVec2D.cpp file. Put these functions in the same order as in the CartVec2D.h file.

a. Implement two constructors: a default constructor for initialising the x and y coordinate both to 0 and a constructor for initialising x and y to the input parameters of the constructor. Use C++11 constructor delegation, see [C++, p. 1180-1181].

b. Implement a setter and a getter function for `x_` and `y_`. These functions must be inline functions [C++, p. 517]. Strive for const-correctness, see [C++ p.537].

c. Overload the output stream operator >>. The output should have the following format, for example: `[2.5, 3.8]`. This function must be a friend function, you must not use `CartVec2D::` for this function. Add the implementation of friend functions at the top of the CartVec2D.cpp file.

```
std::ostream& operator<<(std::ostream &os, const CartVec2D &rhs)
{
   os << ……

   return os;
}
```

d. Overload the `+=` and the `-=` operator as class member functions [C++, p. 587-588]. Code example for `+=` in CartVec2D.cpp:

```
CartVec2D &CartVec2D::operator+=(const CartVec2D &rhs)
{
   x_ += rhs.x_;
   y_ += rhs.y_;

   return *this;
}
```

It is also possible to implement this function inline in the header file.

Coding guideline: a binary operator should always be overloaded as a class friend function, not as a class member function.

e. Overload the binary – operator by using the already implemented `operator-=` class member function.

Code example showing the operator+ in CartVec2D.cpp (class friend function) using the += operator:

```
CartVec2D operator+(const CartVec2D &lhs, const CartVec2D &rhs)
{
   CartVec2D result;
   result += lhs;
   result += rhs;

   return result;
}
```

f. Implement more vector arithmetic:

- the unary – operator (used with one operand, minus sign operator) for changing the sign of x_ and y_, see [C++, p. 601].
- the scalar multiplication operator * for multiplying both x_ and y_ with a double value (scalar value). Two overloads are necessary (see the next code example).

```
// Using C++11 uniform initialisation: {. . .}
CartVec2D cv1{-2.0, 1.0};

// Scalar multiplication
cout << cv1 * 3.0 << endl;     // [-6.0, 3.0]
cout << 2.0 * cv1 << endl;     // [-4.0, 2.0]


// Unary - operator.
cout << -cv1 << endl;          // [2.0,-1.0]
```

g. Test all functions in main(), using several different operators in the same expression.

C++ Assignment 7. Implement the DOFxyRz class

A Roomba vacuum cleaner robot moves autonomous over a flat floor in x and y world coordinates. Besides translating a Roomba can rotate around z (in Roomba coordinate system): Rz. It can even rotate without translating. The z-axis is always perpendicular to the Roomba robot and its origin is in the middle of the moving Roomba.

A Roomba has 3 Degrees of Freedom (DoF): x, y and Rz.

An angle can be expressed in degrees and radians:

- π radians = 180°
- 2π radians = 360°

Using radians is necessary for the input of mathematical functions s in C++, for example, sin() and cos().

The class DOFxyRz contains the following data members: a CartVec2D object (composition relation, has-a relation, see [C++, p. 788]) and a double for representing Rz.

a. Implement a default constructor for initialising x_ and y_ to 0 and a constructor for initialising x_ and y_. Use C++11 constructor delegation.

b. Implement the output stream operator<< for showing the CartVec2D and Rz value. Use the already implemented output stream operator for CartVec2D.

c. Implement a setter and getter for `Rz_`.

d. Overload the operators `+=`, `+`, `-=` and `-` operator- function for doing only vector math with the CartVec2D class data member (not `Rz_`). The rhs operand is typed CartVec2D. The return type is DOFxyRz.

e. Overload the operator+ and operator- function for adding a double value to the Rz_ class data member. The resulted value should remain between -π and π. Use a while loop to increment or decrement the `Rz_` value until this value is in the required range.

f. Implement the member function `CartVec2D heading()const` for calculation the heading vector: x equals cos(Rz_) and y equals sin(Rz_). The returned heading vector must always be sized 1. Use the `length()` function for dividing these values.

g. Test all the implemented class in main(). Show the results with some text.

In the next assignment, we are going to use this class for doing some basic Roomba movement (translation and rotation) simulation. The simulation code is already available.

C++ Assignment 8. Using the DOFxyRz class for simulating Roomba movements

The next code example shows us how we can use the class DOFxyRz for simulating the movements of a Roomba for a constant translation and rotation speed. A simulation is always executed in time steps.

Starting point for the simulation:

- Translation speed: 0.2m/sec
- Rotation speed: 0.25π rad/sec

a. Reed the next code example line by line and find out the simulation goal and the simulation steps taken. This next code example is only compilable if all overloaded operator functions are implemented.

```
// Avoid the usage of #define
const double PI{3.14159265358979323846};

const double DELTA_t_sec{0.1};
const double MAX_t_sec{10.0};

double velocity{0.2};  // m/sec
double angular_velocity{0.25 * PI};  // rad/sec
double t_sec = 0.0;

DOFxyRz roomba;

// Simulation loop
while (t_sec <= MAX_t_sec)
{
   cout << "t = " << t_sec << "  " << roomba << endl;

   // Calculate rotation every simulation step
   roomba += angular_velocity * DELTA_t_sec;

   // Calculate translation every simulation step
   CartVec2D delta_heading(roomba.heading() * velocity * DELTA_t_sec);
   roomba += delta_heading;

   t_sec += DELTA_t_sec;
}
```

The simulation time step `DELTA_t_sec` must be sufficient small to come close to the real movement. If we take the simulation time step too small it will take too much unnecessary simulation time.

b. Write `t_sec` and the resulting x and y values to a CSV (Comma Separated Values) text file. Use a C++ file stream. Put every combination of a time point and the x and y value in one line separated by commas. Do not write any additional characters and texts in these lines to avoid reading problems.

c. What is the shape of the simulated movement?

d. Try different values for example such as 0.0001, 0.001, 0.01 and 0.1 seconds for the simulation time step `DELTA_t_sec`. Compare the results.

You can use a spreadsheet program like LibreOffice or other tooling like GNUplot for reading this CSV text file and creating a 2D plotted diagram. In LibreOffice choose Insert, Chart, XY (Scatter) and select the X and Y axis data column in the spreadsheet.

In the directory Plotting, a Gnuplot script file `plotRoombaPath.gp` is already available and will create a .png file depicting the XY-plot of the Roomba path (trajectory). This script must be made executable and be started in a bash shell:

```
./plotRoombaPath <cvs data file name>.
```

The .png file is created in the current directory. Gnuplot must be installed before we can use the script.

e. Which simulation time step is small enough to get close enough to reality?

C++ Assignment 9. Implement the Device abstract base class and a derived class

An embedded application must control several different connected devices, for example, motors, displays and leds. Every device must support an initialisation, resetting and shutdown function and must be identified by a string containing a unique informative name (the developer is responsible for doing this). This identifier can be used for debugging and logging purposes.

a. We must implement an abstract base class (ABC) for different devices. A base class serve top define a common interface to be used by derived classes, see [C++, p.746-757, 778-779].

!	An abstract base class must have a virtual destructor. If the destructor is not virtual, then just the destructor corresponding to the pointer type is called.

We cannot create objects for an abstract base class. Every concrete derived class must implement all the virtual abstract functions. Otherwise we cannot instantiate any objects.

The abstract base class Device is as follows (using Doxygen formatted comments):

```
/// The class Device is an abstract base class containing all
/// basic device interface functions.
/// This abstract class cannot be instantiated, because this type of
/// classes contains a least one abstract member function.
class Device
{
public:
   Device(const std::string &id);
   virtual ~Device() = default;

   const std::string &getID() const;
   /// Abstract functions, must be implemented in the derived classes.
   virtual void initialise() = 0;
   virtual void reset() = 0;
   virtual void shutdown() = 0;

private:
   const std::string ID_;
};
```

b. Derive the class Motor from the abstract base class Device. The Motor constructor must invoke the Device constructor in its initialization list. So an ID string must also be provided to the constructor of Motor. In all three mentioned virtual functions the RPM value must be set to 0 in the class Motor.

It is possible to use the already implemented RotationMotor class. This class must be updated to support the necessary member functions.

c. Make Device not copy-able because we must force the developer to instantiate every Device object with its own unique id. We cannot change an id by a class member function because it is a const data member.

```
/// The class Motor is_a Device.
class Motor: public Device
{
public:
   Motor(const std::string &id);
   virtual ~Motor();

   virtual void initialise() override;  // C++11 override specifier
   virtual void reset() override;
   virtual void shutdown() override;

   void setSpeed(double speed);
   double getSpeed() const;

private:
   double speed_;
};
```

d. Implement the class member functions for setting and getting the speed value. Instantiate 3 motor objects with unique IDs and show the results from setting and getting in main().

e. We must not use a default destructor for Motor because we must set the speed of the motor to 0 in the destructor by invoking `setRPM()`. This is necessary if we are going to use real interfacing hardware that will store the speed in some hardware register.

f. Print in every implemented virtual function a text to cout containing information what this function is doing.

C++ Assignment 10. Implement the NullDevice class and a polymorphic vector

We are going to implement the **Null Object Pattern** for a derived class from Device. The null object pattern is a design pattern that simplifies the use of dependencies that can be undefined. We want to avoid the checking of a null pointer value. This is achieved by using instances of a concrete class that implements a known interface (doing nothing), instead of null pointers.

A second application is if we want to see quickly whether certain parts of the software are already functioning properly if derived classes or the hardware are not yet available. We create a derived class NullDevice in which we implement all virtual functions doing nothing or displaying an informative text. In many cases we can write these texts to a log-file or logger (if available).

!	C++ coding guideline: do not use C++03 NULL pointer any longer, but the C++11: `nullptr`.

a. Create 2 Motor and 3 NullDevice objects with unique IDs and fill the polymorphic `vector<Device*>` with the addresses of these objects. The compiler allows to place the addresses of objects of different derived types in a pointer type to the base class.

   This mechanism is called late binding, see [C++, p.737]. The compiler generates code that allows the correct virtual function to be selected at run-time.

b. Use a range-based for loop for traversing this polymorphic vector and invoking the `getID()` and `reset()` functions by using the pointers contained in the vector. Show the IDs using cout.

Optional C++ Assignment 11. Implement CartVec2D as a class template

Typing of the x_ and y_ coordinate is possible by a class template parameter T at compile-time, see [C++, p.830-836].

The next two notations can be used for template classes:

```
template <class T>
class CartVec2D
{

template <typename T>
class CartVec2D
{
```

```
/// The class CartVec2D represents a Cartesian coordinate in 2D.
template <typename T>
class CartVec2D
{
public:
   …

private:
  T x_;
  T y_;
};
```

a. All class template member functions and friend functions must be implemented in the header file, see [C++, p. 831-837].

b. Instantiate CartVec2D objects for the next C++11 types: `double`, `int8_t`, `int16_t` and `int_fast8_t`. The types with known standardised byte size play an important role in (low level) embedded systems software.

   Create some (complex) expressions using this template class and print all resulting values to `cout`.

   Is it possible to use mixed types in these expressions?

---

3. UML assignments

Use plantUML for creating the UML diagrams.

UML Assignment 1. Class diagram for Dataframe (C++ Assignment 4)

Draw the *class diagram* for the Dataframe class showing all data and function members, and their visibility.

UML Assignment 2. Class diagram for CommandProcessor (C++ Assignment 5)

Draw the *class diagram* for the CommandProcessor class showing all data and function members, and their visibility.
