Mobile Robot Control 2020: Tutorial Lecture #1 Implementation

APRIL 29TH 2020

Jordy Senden, Hao Liang Chen, Wouter Kuijpers

Department of Mechanical Engineering, Control Systems Technology, Robotics

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Contents

- In this part, we cover various topics ("*lessons learned from tutors*")

- (Try to) adopt a top-down approach:

  how to structure your code

  to

  how to implement functionality.

Attention: does not replace the C++ tutorials, but shows you how to apply them to development for Robotics!

main function

- The software starts execution of the main-function.

- (very) high-level overview main-function:
  1. initialize (variables, hardware, software, etc)
     `emc::Rate r(EXECUTION_RATE);`
     `emc::IO io`
  2. loop as long as task can be executed
     `while(io.ok())`

main.cpp (short)

```
#include <emc/io.h>
#include <emc/rate.h>
       ...

int main(int argc, char *argv[])
{
    // initialization
    emc::Rate r(EXECUTION_RATE);
    emc::IO io;
       ...

    // while-loop
    while(io.ok()) {
       ...

       r.sleep();
    }
    return 0;
}
```

Software Pattern

- The loop will contain a lot of functionality!

  Requires: structure (e.g. a software pattern)

Event Loop (pseudo-code) (from: *Do's and Don'ts in the design of a robotic software architecture" by Herman Bruyninckx*)

```
when triggered     // by operating system
do {
     communicate() // get data from other activities
     coordinate()  // decide what phase of plan to switch to
     configure()   // set all parameters and select functions
     compute()     // execute control, perception, monitoring, plan
                   // functions synchronously, one after the other
     coordinate()  //
     communicate() // send data to other activities
     sleep()       // the loop deactivates itself, until next deadline
}
```

!

Assumption:
single process

Software Pattern

- We provide you with `sleep()` but you are free to expand to whatever!

Event Loop (code)

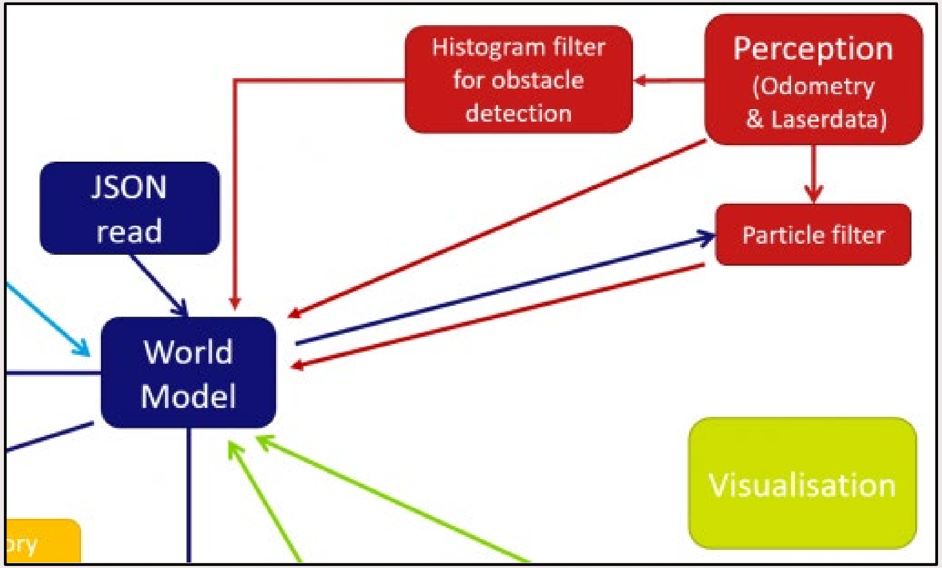
```
// initialization
emc::Rate r(1); // [Hz]

while(io.ok()) {  // periodic triggering i.c.w. sleep()-function
    coordinate()
    configure()
    compute()
    r.sleep();    // sleep until new periodic trigger
}
```

- This is readable! But how to keep it this way?

Components

- Your software will consist of **components** and **interfaces** between the components.

- How to use tools from C++ to embed this architecture in your code?


Histogram filter for obstacle detection
Perception (Odometry & Laserdata)
JSON read
Particle filter
World Model
Visualisation
ory

(part of) the software architecture of Group 7 from EMC2019.

Tutors' Tip

How to separate Components

```
int main(...)
{
    while(io.ok()) {
       <snippet 1>
       <snippet 2>
       r.sleep();
    }
    return 0;
}
```

No Functions

Readability

Maintainability

```
int main(...)
{
    while(io.ok()) {
       Snip1();
       Snip2();
       r.sleep();
    }
    return 0;
}

void Snip1(){
    <snippet 1>
}
```

Functions

Readability

Maintainability

```
int main(...)
{
    while(io.ok()) {
       Snip1();
       Snip2();
       r.sleep();
    }
    return 0;
}
```

```
void Snip1(){
    <snippet 1>
}
```

Functions in Files

Readability

Maintainability (e.g. Git)

```
int main(...)
{
    class1 Test;
    class2 Drive;
    while(io.ok()) {
       Test.Snip1();
       Drive.Snip2();
       r.sleep();
    }
    return 0;
}
```

```
class class1{
    void Snip1(){
       <snippet 1>
    }
}
```

Classes →

Classes

- Here we continue with classes, due to some extra benefits.
  - a class can contain functions (called methods)

    `driveForward(...);`
  - a class can contain variables

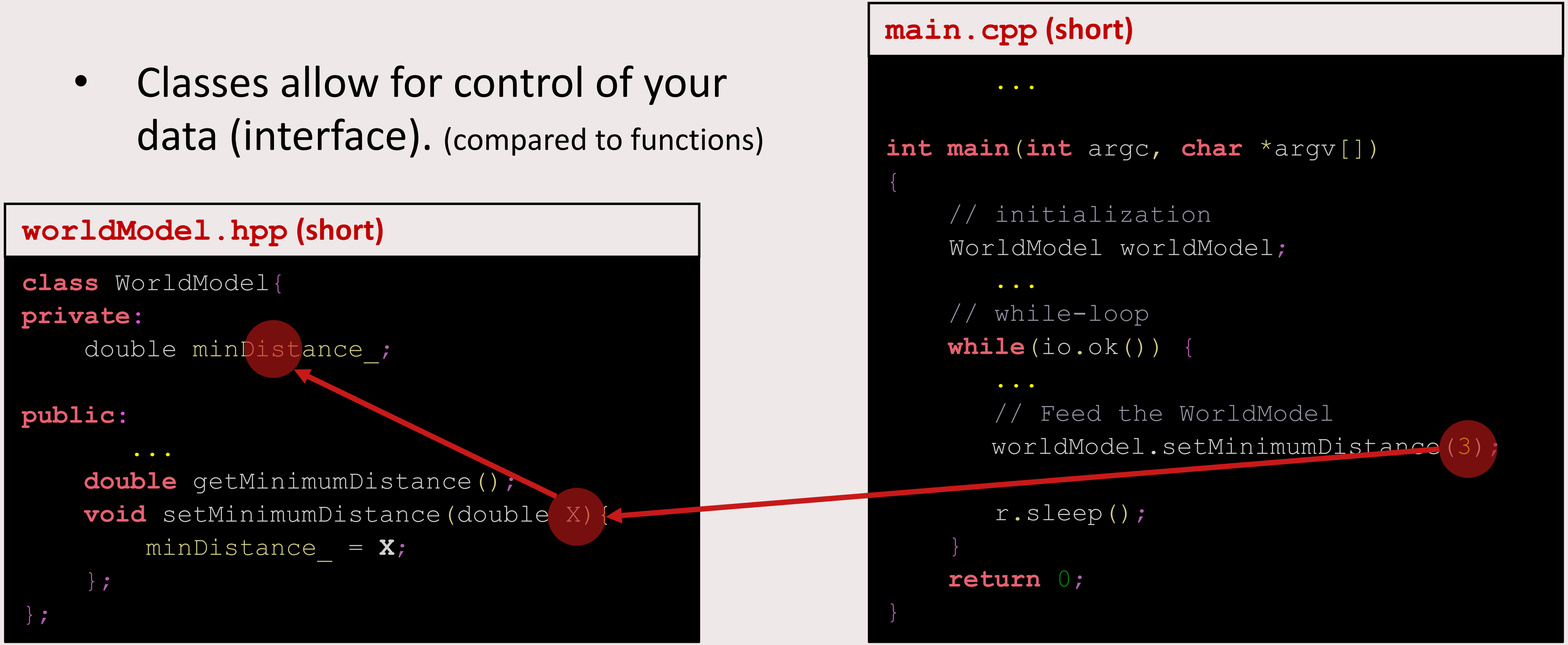
    `odom;`
- These benefits can be used for implementing interfaces between software components.

driveControl.hpp (short)

```
#include <emc/io.h>

#include <emc/odom.h>


class DriveControl{
private:
    emc::IO *inOut;
    emc::OdometryData odom; // [x,y,a]

public:
    DriveControl(emc::IO *io){
        inOut = io;
        odom = emc::OdometryData();
        return;
    }

    void driveForward(double Xspeed);
    double driveBackward(double Xspeed);
    double rotate(double Aspeed);
    void stop();
};
```

Interfaces (set)

- Classes allow for control of your data (interface). (compared to functions)



worldModel.hpp (short)

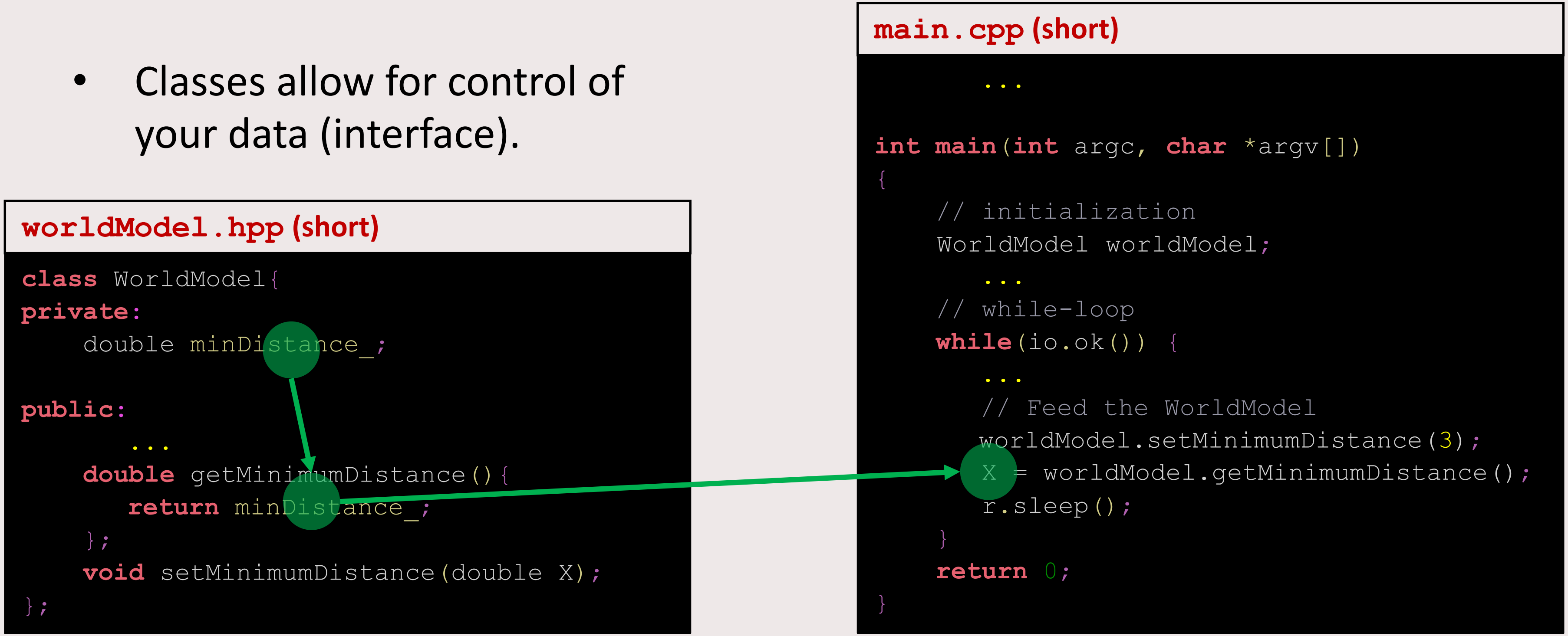
```
class WorldModel{
private:
    double minDistance_;

public:
        ...
    double getMinimumDistance();
    void setMinimumDistance(double X){
        minDistance_ = X;
    };
};
```

main.cpp (short)

```
       ...

int main(int argc, char *argv[])
{
    // initialization
    WorldModel worldModel;
       ...
    // while-loop
    while(io.ok()) {
       ...
       // Feed the WorldModel
       worldModel.setMinimumDistance(3);

       r.sleep();
    }
    return 0;
}
```

Interfaces (get)

- Classes allow for control of your data (interface).



worldModel.hpp (short)

```
class WorldModel{
private:
    double minDistance_;

public:
        ...
    double getMinimumDistance(){
       return minDistance_;
    };
    void setMinimumDistance(double X);
};
```

main.cpp (short)

```
        ...

int main(int argc, char *argv[])
{
    // initialization
    WorldModel worldModel;
        ...
    // while-loop
    while(io.ok()) {
        ...
       // Feed the WorldModel
       worldModel.setMinimumDistance(3);
       X = worldModel.getMinimumDistance();
       r.sleep();
    }
    return 0;
}
```

Structures & Enumerations

- Use the extra semantics of advanced variables to improve your code, e.g. a wall.

- Enumerations can only take a set of values, e.g. `color`.

example.cpp (short)

```
struct wall {
  point2D startPoint;
  point2D endPoint;
  color   wallColor;
};

struct point2D {
  double x;
  double y;
};

typedef enum{
  red,
  yellow,
  green
} color;
```

Alternative: array
Issue: semantics of elements

Alternative: double
Issue: non allowed entries (e.g. $-\pi$)

Prevent Blocking Functions!

- **Rule:** a function in the `while`-loop should not block execution.
  - e.g. if `doDifficultStuff()` would halt execution (using while-loop) until a wall appears, `runEverySecond()` will not run every second!

- **Rule:** a function in the `while`-loop should not take to long to compute.

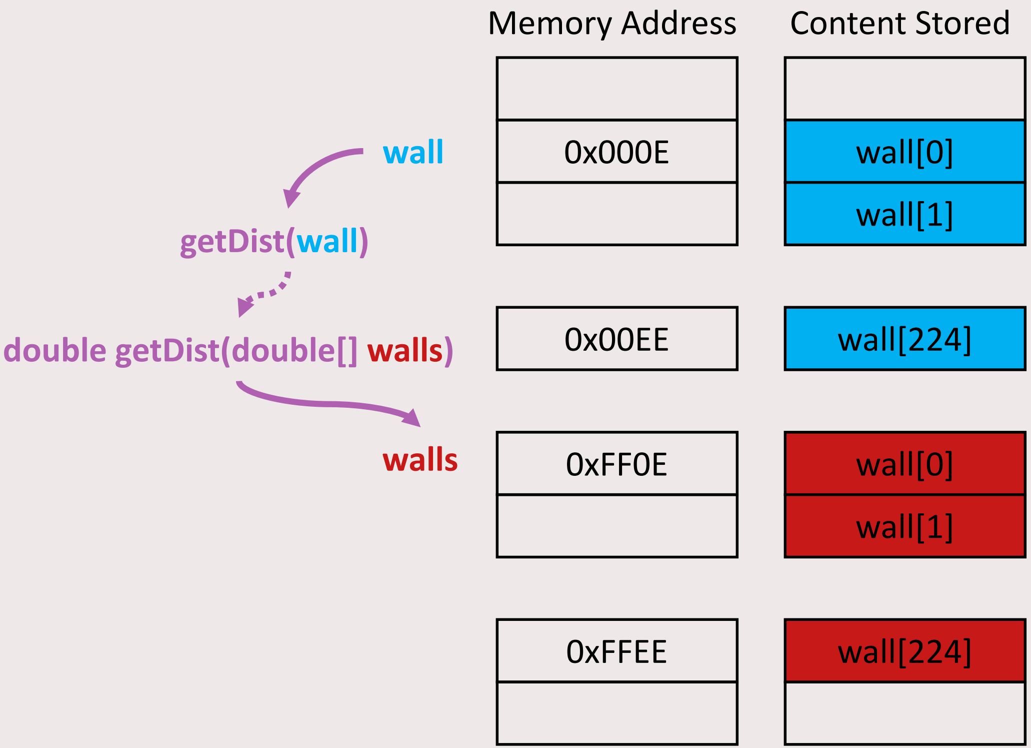
main.cpp (short)

```cpp
#include <emc/io.h>
#include <emc/rate.h>
       ...

int main(int argc, char *argv[])
{
    // initialization
    emc::Rate r(1); // [Hz]
    emc::IO io;
       ...

    // while-loop
    while(io.ok()) {
       doDifficultStuff();
       runEverySecond();
       ...

       r.sleep();
    }
    return 0;
}
```

Pointers

- A variable with the value of an memory address.

- Example: variables input to functions are copied to a new memory address.
  - scalars: mwah
  - array of laserdata: lot of copies!



wall

getDist(wall)

double getDist(double[] walls)

walls

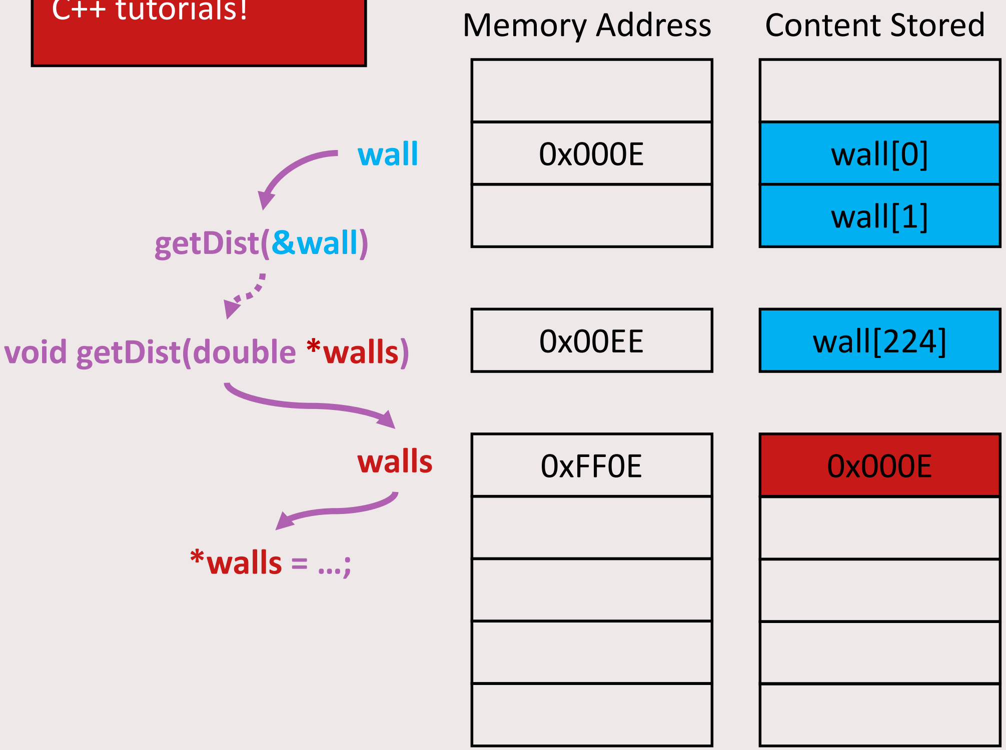
Memory Address	Content Stored
0x000E	wall[0]
	wall[1]
0x00EE	wall[224]
0xFF0E	wall[0]
	wall[1]
0xFFEE	wall[224]

Pointers

Attention: see the C++ tutorials!

- A variable with the value of an memory address.
- Example: variables input to functions are copied to a new memory address.
  - scalars: mwah
  - array of laserdata: lot of copies!

→ especially when changing original data



```
wall
getDist(&wall)
void getDist(double *walls)
walls
*walls = …;
```

Memory Address	Content Stored
0x000E	wall[0]
	wall[1]
0x00EE	wall[224]
0xFF0E	0x000E

Magic Numbers

- values (no variables) with unexplained meaning
- avoid using values, see e.g. `config.h` in the example

`main.cpp` (short)

```
      ...

if(fabs(distanceBackwards) >= 0.1)
{
    // we start rotating
    state = rotate;
}
```

The 0.1 gives no information

`main.cpp` (short)

```
ROBOT_RADIUS = 0.1 // [m]

if(fabs(distanceBackwards) >= ROBOT_RADIUS)
{
    // we start rotating
    state = rotate;
}
```

Better!

!

Attention: why ROBOT_RADIUS?