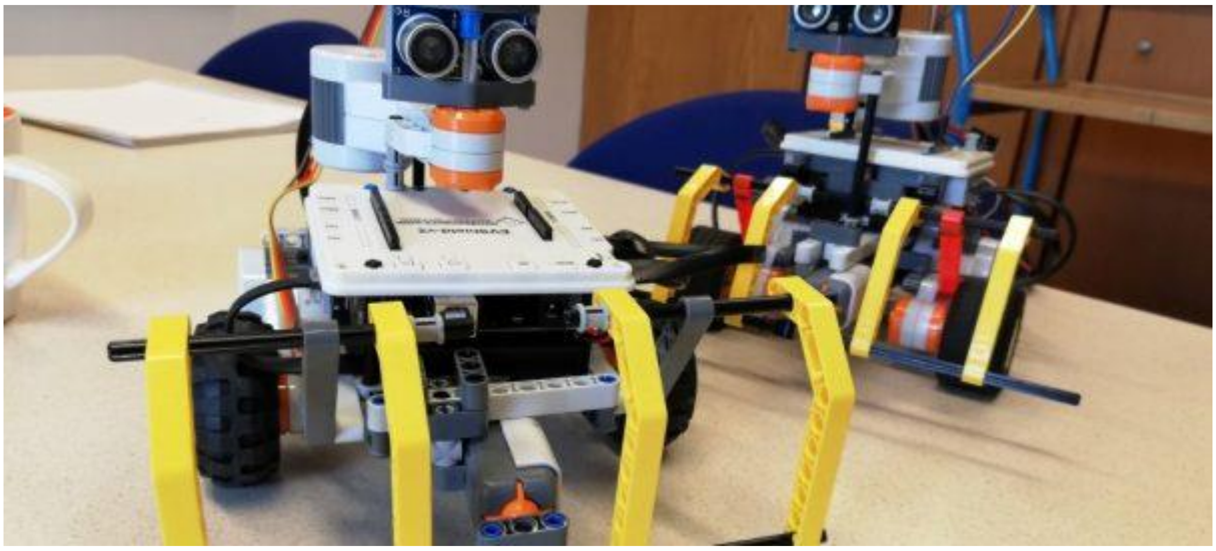


Assignment 4a

Content Application Development Day 4
<u>Lecture</u>
In the lecture you will learn more about programming of the Explorer robot in the C++ programming language using the Arduino IDE. Topics "Choices and repetitions" as well as repetition statements such as *for* and *while* were discussed in the previous lecture.
<u>Tutorial</u>
In the tutorial you will practice building a controller for a mobile robot built with Lego & EVShield.

At the end of day 4 you will be able to
- Build a controller for a mobile robot based using an object-oriented design
- Use and apply sensors and actuators in a program.
- Use if statements to add conditional execution to a program.
- Understand some of the differences between Java and C++

Reading between day 4 and day 5
<u>Web:</u> Study Arduino Tutorial and EVShield motor commands.
<u>Book:</u> From “Head First Java” (K. Sierra & B. Bates): p10-13, 4-5, p240-249 and from the book *Aan de slag met Java [Getting Started with Java]:* Chapter 5*.*

Extra exercise
Try to complete the challenges of Practical Session 2. It will be of great help for this assignment.

Who is this assignment for?
This assignment is intended for those who participate in the SmartProducts project and who have a Lego Mindstorms kit.

For this assignment it is useful for the project group to get together and to do the assignment in pairs. Two pairs can then share a robot and alternately test their programs.

What do you need for this assignment?

- Arduino IDE with libraries as given for this course
- A mobile robot
- Battery pack with full batteries
- EVShield documentation
- Example Arduino project "explorer1"

Setup of mobile robot 'explorer'

Today, we are going to make a mobile robot that can independently move through a space, avoiding any obstacles.
An example of a robot you can use for this is the Explorer, the construction guide for this robot can be found here. However, you are free to use a different robot.
In this assignment we use these sensors:

- An Ultrasonic sensor that has been mounted onto a motor. This enables the robot to look around for a way out if the robot encounters an obstacle.
- A Touch sensor behind a bumper, which will detect if the robot bumps into something.

Connections
The motors and sensors are connected to the EVShield as follows (**check these!**):

- 2 motors driving the wheels connected to M1 and M2 of Bank A.
- 1 motor with the Ultrasonic sensor connected to M1 of Bank B.
- Touch sensor connected to BBS2 (Bank B)
- Ultrasonic sensor: TRIGGER: Digital pin 3 (at servo header), ECHO: Digital pin 5, GND, Vcc to 5V, as detailed here.

Parameters
Where to find some of the parameters, which might have to be adapted to your version of the robot. Check these!:

Explorer.h:

- Wheel diameter and Trackwidth, class variables: initial direction that is 'forward', motor on which Ultrasonic sensor is mounted.
- Ultrasonic sensor and Touch sensor details.

Contents

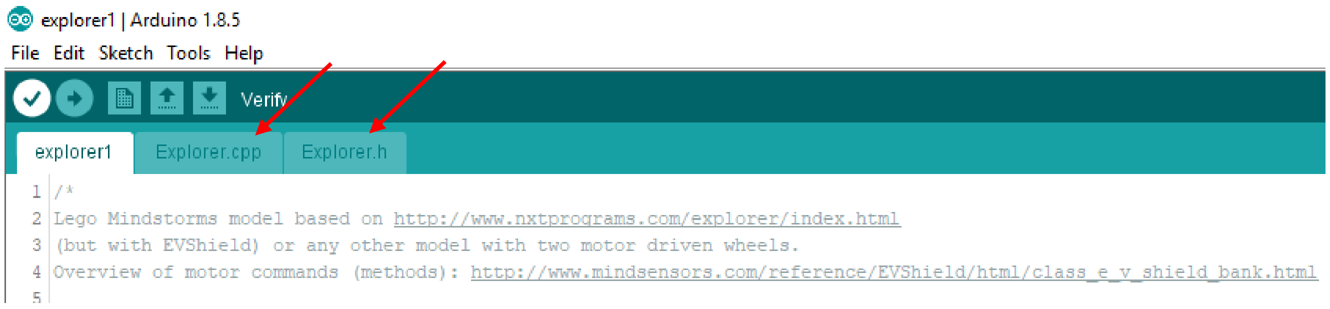
1. Start with Arduino project using given code

Download the example project "explorer1" and extract it to the Arduino folder (Documents\Arduino).

This is an Arduino project that should work out-of-the-box: upload it to the Arduino and test:

If you press the Go-button the Explorer robot should start driving. Pressing the Go-button again should stop the robot.

The Arduino project has an added **class 'Explorer'**, which is declared in the header file (Explorer.h) and its methods are defined in the C++ file (Explorer.cpp):



```
/*
Lego Mindstorms model based on http://www.nxtprograms.com/explorer/index.html
(but with EVShield) or any other model with two motor driven wheels.
Overview of motor commands (methods): http://www.mindsensors.com/reference/EVShield/html/class_e_v_shield_bank.html
```

Check out the class definition so you know where to find its methods. Also, carefully read the next paragraph so you understand how and why this class was designed.

2. Designing the Explorer

Most of the class design has been done for you: the skeleton of the class and its variables is already there. How did the design of this class evolve? First, a name for the class was established and some methods where identified:

```
class Explorer {

  drive();
  stop();
  turn();
}
```

Identifying methods
Ask questions like:
What can the robot do?
(check slides of session #2 for more info on designing software)

Next, some thought was put into how the Explorer could sense its environment by using an ultrasonic sensor (Sonar), and a touch sensor (behind the bumper). Also parts the robot consists of where identified (motors) and how these will be controlled (with an EVShield). Because the EVShield will manage the motors, at this stage it seems not necessary to include the motors themselves into the design:

```
class Explorer {
  evshield;
  sonar;
  touch;

  drive();
  stop();
  turn();
}
```

Identifying class-variables
Ask questions like:
What does the robot consist of?
Or
What are its working principles?
(check slides of session #2)

During the detailed design, you could perform small tests/try things, to establish knowledge about use of libraries, how to do … (eg. how to read a sensor, or how to turn on a motor), get parameters etc.

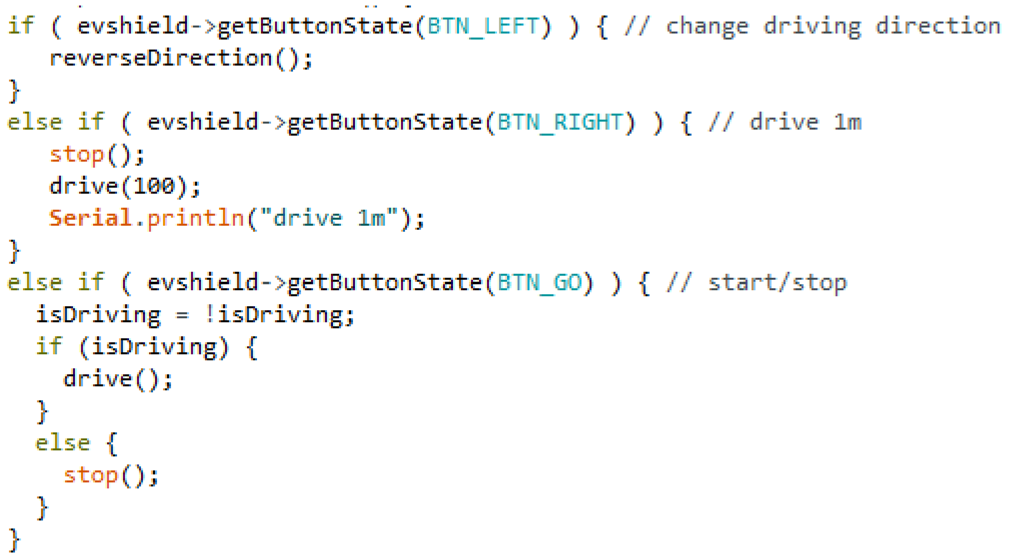
```
class Explorer {
  EVShield * evshield;
  NewPing * sonar;
  EVs_NXTTouch * touch;

  bool isDriving = false;
  bool forward = true;
  SH_Direction motor_direction = SH_Direction_Reverse;
  int speed = SH_Speed_Slow;
  SH_Motor sensorMotor = SH_Motor_1;

  void drive(distance);
  void stop();
  void turn(angle);

  void checkSensors();
}
```

Identifying class-variables and their values, parameters of methods
Ask questions like:
What are the specifications?
Or
How … should it drive/stop/turn?
(check slides of session #2)

This paragraph just highlighted some parts of the design process of the class Explorer, but it is not complete. Some more methods where added later. Do you understand what these methods do?

3. Reading sensors and responding to these

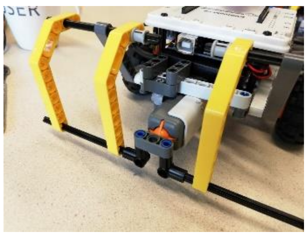
In the final design phase, a method was added which should take care of all sensors: **`checkSensors()`**. The buttons on the EVShield are considered to be sensors also. Part of this method is already filled in:



```
if ( evshield->getButtonState(BTN_LEFT) ) { // change driving direction
   reverseDirection();
}
else if ( evshield->getButtonState(BTN_RIGHT) ) { // drive 1m
   stop();
   drive(100);
   Serial.println("drive 1m");
}
else if ( evshield->getButtonState(BTN_GO) ) { // start/stop
  isDriving = !isDriving;
  if (isDriving) {
    drive();
  }
  else {
    stop();
  }
}
```

Check if the explorer behaves like described here. Eg. if the Go-button on the EVShield is pressed, does it start driving? And when pressed again, does it stop? Also, do the other buttons work?

4. Respond to touch sensor: bumper was hit?



Add an if-statement to the method **`checkSensors()`** that detects if the touch sensor was hit. This should be done only when driving in a forward direction. So first we should check that: are we driving in a forward direction? Add this if-statement to check that to the method **`checkSensors()`**:

```
if (isDriving && forward) {
}
```

Inside this if-statement, we can check if the touch sensor was pressed (you have used this before, look it up). Hint: use the **`isPressed()`** method of the object **`touch`**. There is a small 'catch' here: The Explorer class uses object-references (pointers) instead of the object itself. If referencing a method or class variable of an object-reference, we should use an arrow **`->`** instead of a dot. So calling method **`isPressed()`**of the object-reference **`touch`** in code looks like this:

```
touch->isPressed()
```

So, the complete nested if-statement looks like:

```
if (isDriving && forward) {
  if (touch->isPressed()) {
       Serial.println("bump");
       // call method to reverse & turn
  }
}
```

In the comment we have already put a hint on what should happen: "call method to reverse & turn". This method will perform the reverse & turn and must be added to the class Explorer. You will first have to add its declaration to Explorer.h:

```
void reverseTurn();
```

And then add the method definition to Explorer.cpp:

```
void Explorer::reverseTurn() {
  /* Stop, drive backward a little and then make a turn,
  to 'escape' from something if we bumped into something */
  Serial.println("reverseTurn()");
}
```

To help testing, we have added a print-statement. Now you may add calls to the other explorer methods which will make the robot stop, drive backward a little (eg. 30 cm) and turn. You can call the methods `stop()`, `drive()` and `turn()` of the Explorer to realize this. Read the comments at those methods to see how to use them. For instance, to drive backward a certain distance, the `drive()` method needs a negative parameter.

When the method is ready, call it from the if statement in the method **`checkSensors()`**.

Now test if the bumper works.

5 Add behaviour to avoid obstacles

The ultrasonic sensor can be used to detect obstacles. If an obstacle is detected, the robot can ‘look around’ to find a way out. Extend the previously added if-statement with an ‘else’:

```
if (isDriving && forward) {
  if (touch->isPressed()) {
       Serial.println("bump");
       // call method to reverse & turn
  }
  else { // check ultrasonic sensor
  }
}
```

In this else-statement, take a reading from the ultrasonic sensor (measure its distance):

```
unsigned int distance = sonar->ping_cm();
```

After that, add an *if*-statement to determine if something was seen: if the distance is larger than 0 and smaller than for instance 30 cm, then an object was detected. If that is the case, the robot must find a way out. Create a method which does this and add it to the Explorer class.

In pseudo code, this method should do something like:

```
Stop.
Turn the Ultrasonic Sensor 90 degrees (use motor on top!)
Read the value of the Ultrasonic Sensor (look left)
Turn the Ultrasonic Sensor -180 degrees
Read the value of the Ultrasonic Sensor (look right)
Turn the Ultrasonic Sensor 90 degrees (faces in original direction)
Compare the two read values of the Ultrasonic Sensor:
If the first value is smaller than last?
        Turn right
Else:
        Turn left
Drive. (Continue moving)
```

How to do this was already discussed in Practical Session 2. Look at the slides of that session if you need additional help. Also use the methods `stop()`, `drive()` and `turn()` of the Explorer when applicable.

6. Show travelled distance

When the robot stops when the Go-button is pressed, we would like to view the distance the robot has travelled.
To retrieve the distance covered, we can get the number of degrees a motor has turned after its last reset (all motors will be reset by the init-method, when the program starts). To retrieve the degrees turned from one of the driving motors, we can use:

```
evshield->bank_a.motorGetEncoderPosition(SH_Motor_1)
```

To calculate the distance travelled, we first determine how many rotations this is (divide by 360) and use the absolute value of the number of degrees:

```
abs(evshield->bank_a.motorGetEncoderPosition(SH_Motor_1))/360
```

Multiply this by the circumference (`WHEEL_DIAM * PI`) and divide by 100 to get the travelled distance in meters. Display this after the call of the stop() method, when the Go-button is pressed.

Print the travelled distance in meters to the Serial Monitor.

Finish

Test your program and give a demonstration to an assistant or lecturer once it is finished.
Requirements for a 'pass' for this assignment:

- The explorer can -while driving around- avoid obstacles by both using its bumper and the mounted ultrasonic sensor
- Two separate methods have been added to the Explorer class: reverseTurn() for the bumper, and another (name chosen yourself) for the ultrasonic sensor.
- Code to show the travelled distance has been added
- Comments and author details have been provided with the code.

Summary

Today you wrote Arduino code for a mobile robot and learned how to control the mobile robot.
In addition, you have learned the following.

- Applying a robot controller using behavior and apply navigation.
- Using and applying Lego sensors.
- Displaying text in the Serial Monitor.
- Analysing data using selection statements.