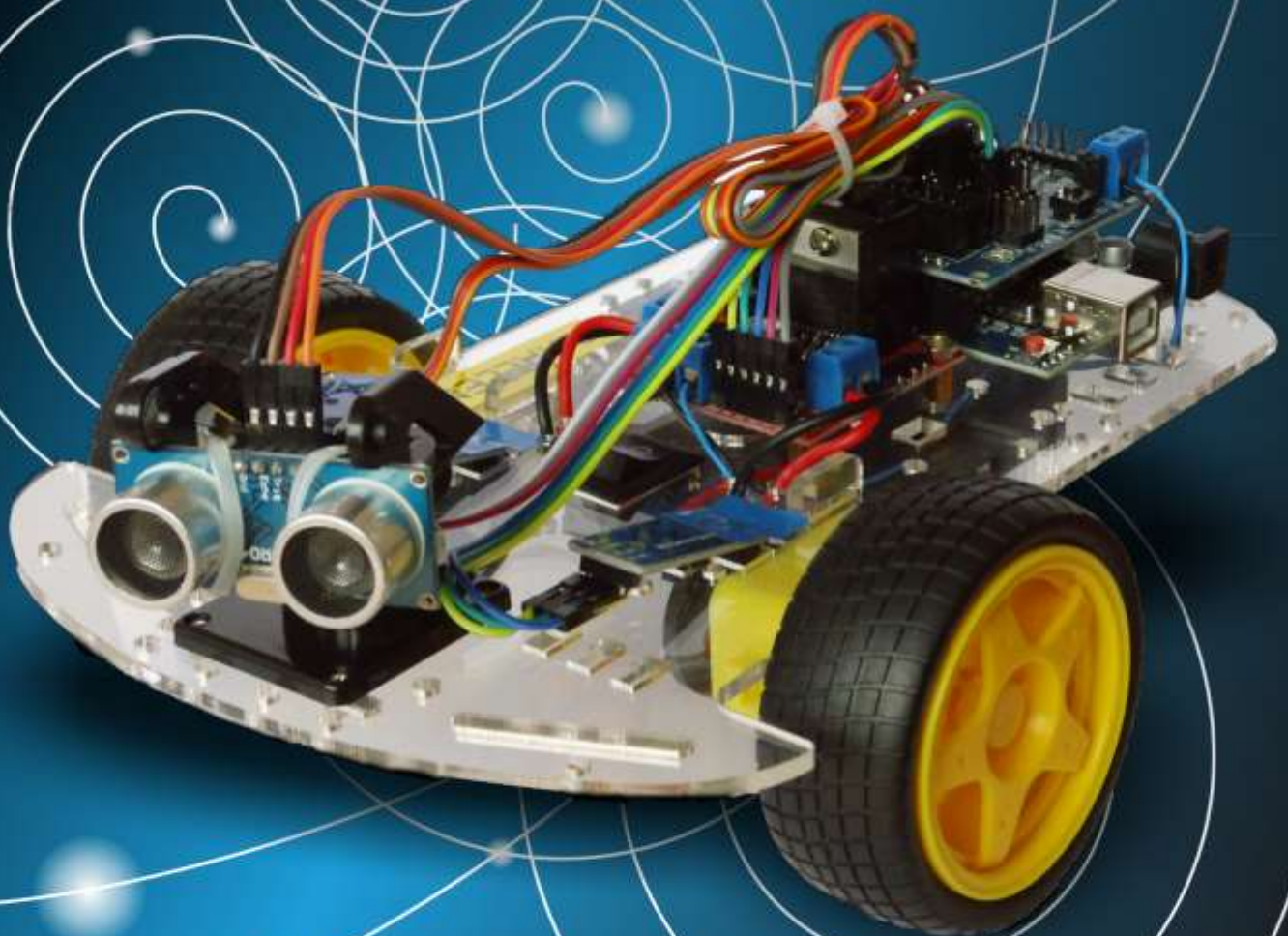


# CHALLENGEBOT

## INSTRUCTIONS



Jaap Daniëls – V1.1 October 2017

## Colophon

ChallengeBot Instructions V1.0

©Jaap Daniëlse 2017

Published under GNU Free Documentation License version 1.3 (GNU FDL)

The Arduino Sketches referred from this document are published under the GPL3 License (shareware).

Both documentation and sources can be downloaded at:

<https://github.com/JaapDanielse/ChallengeBot.git>

## Document history

Ver.	Date	Description	Author
V1.0	20-10-2017	Original version	Jaap Daniëlse
V1.1	22-10-2017	Modified secondary battery case placement.	Jaap Daniëlse

## Introduction

In 2016 I organized a corporate event under the name DevLab where we built robots similar to the one described in this manual. This was a very successful event with over 50 participants. With overwhelming enthusiastic reactions. The preparation of the kits however was a lot of work and not easily portable. So looking for a robot that was better suited. I found this in the kit described here.

### Kit

The main kit is bought as one item but unfortunately not complete enough to build a working robot from. The extra items needed are listed further on. I will ask the supplier if it is possible to create a kit containing all items needed.

When building this robot remember that is not a kit made by a big company that can only be built the right way and tested many times. In this kit the components sort of fit together, but you can definitely build it wrong. When building you are actually engineering your own robot and you have to get it to work.

Also be aware that components may be defective. They are affordable and that means sometimes quality is a bit poor. Over the 50 kit I provided I had about 5 components that were missing or defective. In a few cases I could fix them easily in the other cases the supplier sent replacements.

### Licence

I licensed The documentation under GNU FDL and the Software under GNU GPL3 this means both are available for personal and commercial use and may be used to create your own version and improvements. The Improvements however should be contributed back.

The software is intended as a starting point for your own development and you do not need to contribute this back. But if you improve the core then please do.

I would appreciate it if you let me know how things went.

j.danielse@gmail.com

Jaap Daniëlse



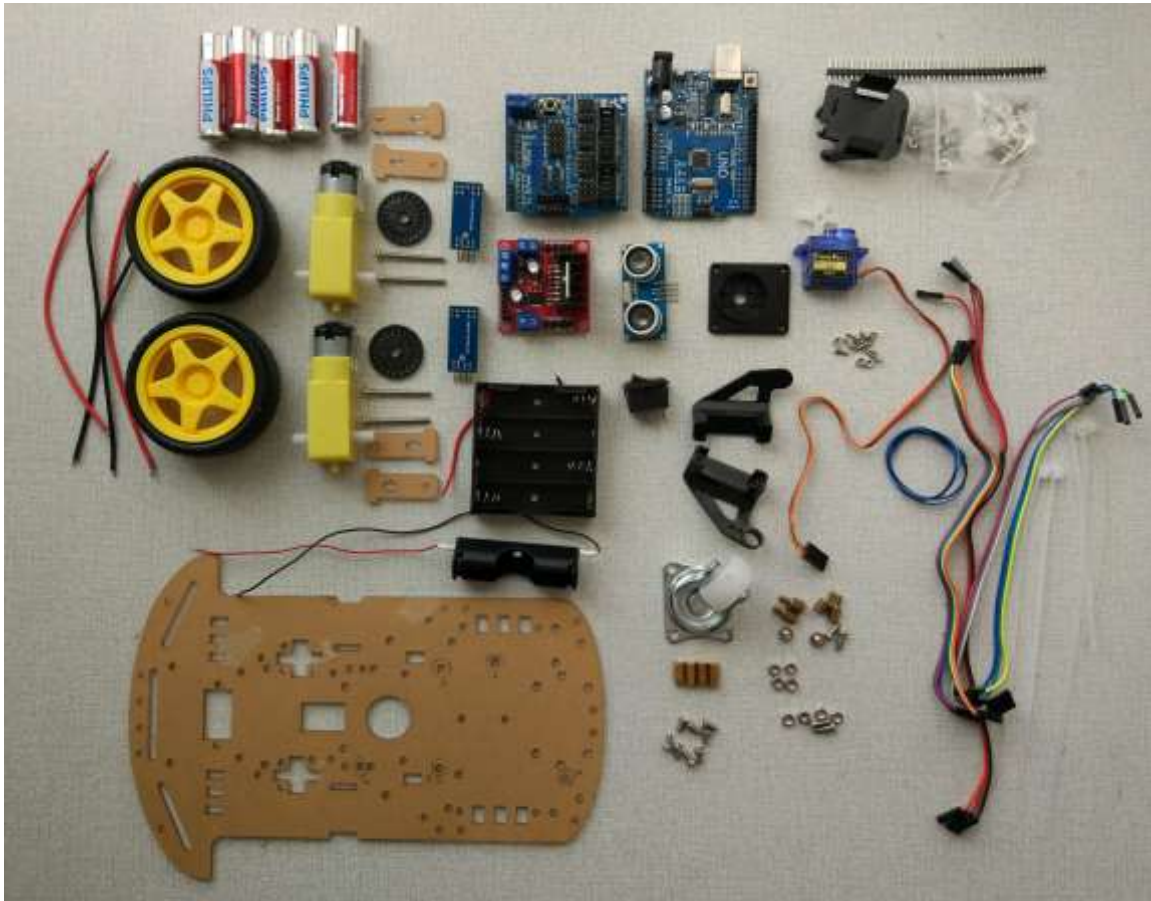
## Components

The robot is based on a kit with chassis, motors, wheels, Arduino processor board, motor board, distance sensor, servo and a battery case. To make the kit complete wires mounting materials and two speed sensors were added.

This makes for a sophisticated robot that can “see” things and know where it is going.

In most available kits the speed sensors are left out making it hard to go straight and know how far you have moved. Using these components the kit is cheap but complete.

An overview of the kit and the main components:



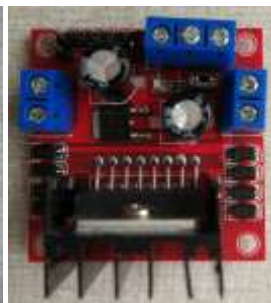
Arduino Uno R3 compatible board



Sensor shield



Speed sensor



Motor board



### Ultrasonic distance sensor

### *The original Robot kit*

Chassis plate  
Motor (2)  
Wheel (2)  
Index wheel (2)  
Motor bracket (4)  
Motor screws (4)  
Motor wire (4) (2 red, 2 black)  
Swivel wheel  
Standoff (4)  
4-battery case  
Switch  
Arduino board  
Usb cable  
Sensor shield  
Motor board  
Ultrasonic distance sensor  
Servo + horns and screws  
Servo bracket + screws  
A set of nuts and screws.

### *Tools*

Soldering iron  
Scissors  
Wire stripper or Knife  
Wire Cutter  
Pliers  
Screwdriver Phillips  
Screwdriver Torx  
Ruler

### *Extra needed for this build*

Speed sensor (2)  
3 wire Dupont cable (20 cm) (2)  
4 wire Dupont cable (30 cm)  
6 wire Dupont cable (20 cm)  
Black power wire (approx. 20 cm)  
Blue power wire (approx. 20 cm)  
Tie wrap (6)  
Standoff M3 (4)  
Screws M3 (8)  
2-battery case  
AA-battery (6)

### *You also need (just small pieces)*

Insulation tape  
Double adhesive tape  
Solder



## Soldering the motors

For connection of the motors 4 wires are supplied (red and black). These wires have to be soldered to the motor.

If the soldering iron is not yet available you can do this later

In case you have forgotten the art of soldering a short instruction.



Strip the wire on both sides using knife or pliers



Twist the stripped end



Tin the stripped, twisted ends



Tin the motor solder lug



Solder the wire to the solder lug

Soldering needs to be done swiftly otherwise plastics may melt and render the motor useless.



The end result should look like this.



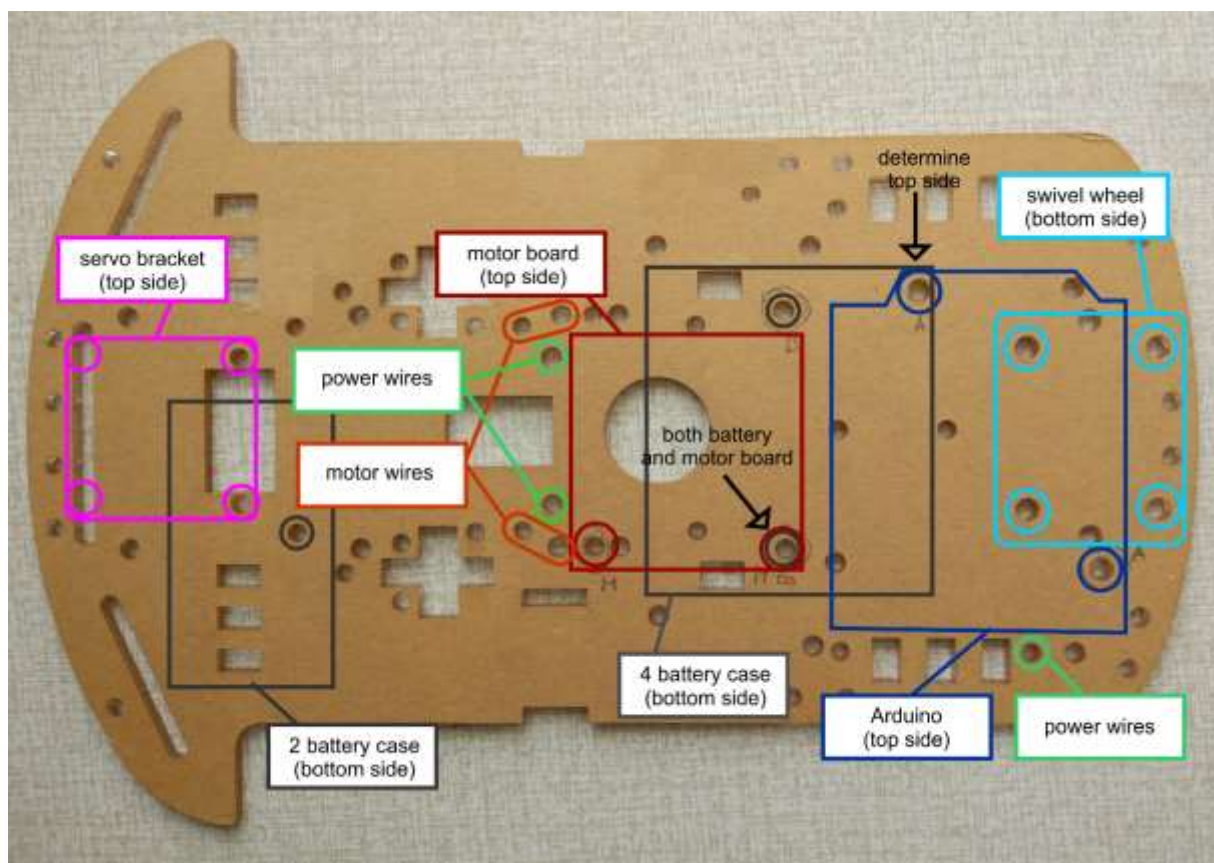
## Assembling the chassis

To begin with... the brown stuff is a protective cover which peels off.



### Find the Top

Second you have to find the top side of the chassis plate. The asymmetric position of the mounting holes for the Arduino board (black arrow) is key here.



This picture also shows you how to identify the holes to use for the screws, standoff's and wire passage.

#### 4-Battery case and Motor board

Because the battery case, which is placed under the chassis, is overlapping the standoff's on which the Arduino board is sitting we have to start with the Arduino standoff's.

The standoff's may be different than the ones shown here with a nut and screw. Standoffs with two screws are more common. The construction however remains the same.



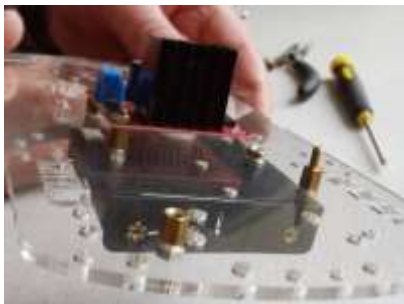
Find the right holes and place the Arduino standoff's



Put two screws in the battery case and fasten them with nuts. These act as spacers.



Now attach the battery case to the bottom side of the chassis. One side with a nut. The other with a standoff.



The standoff is to mount the motor board. This needs a second standoff. Mount this and attach the board.

#### Swivel Wheel

Find the swivel wheel, (long) standoffs and attach to the bottom of the chassis.



Find the components (long standoff's)



Locate the right holes and attach the standoff's on the chassis. (bottom side)



Attach the swivel wheel itself.

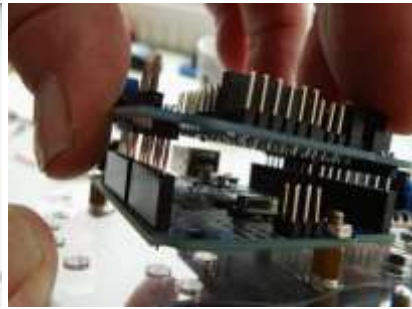


## Arduino Board, Sensor Shield and Switch

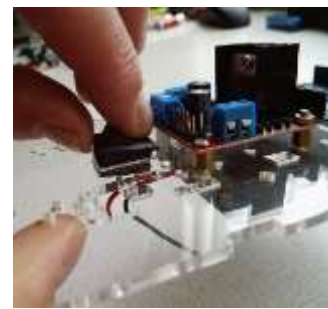
It's time to put in the "brains": the Arduino board.



On the top side the standoff's for the Arduino board are already there. Fasten the board with two screws.



Now carefully set the Sensor shield on top of the Arduino board. Make sure all pins go into the right holes.



Clip in the power switch.

## Servo bracket and Ultrasonic distance sensor

Next is the servo bracket and the ultrasonic sensor. We need to do this now because the second battery case is overlapping the bracket mounting screws.



First you have to make the servo horn fit the bracket. I used a wire clipper and a file.



After some work it will fit nicely.



With the tiny screws that come with the bracket attach the horn to the bracket base.



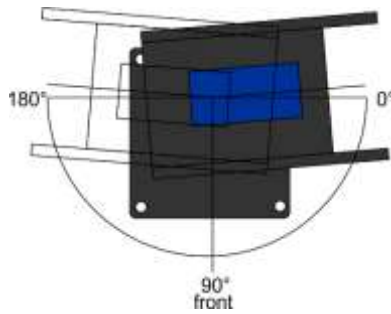
Place the servo in the bracket top.



Fasten with screws.



Place on top of the base



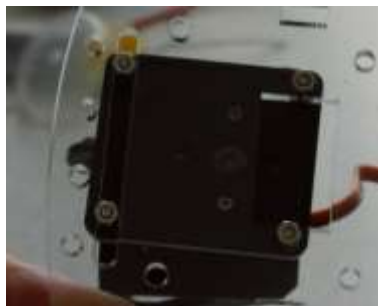
Now gently (without forcing the gears) rock the servo back and forth till it can turn then turn it one way until the end stop.

Remove the base again and place it just past the 0 or 180 degree point.

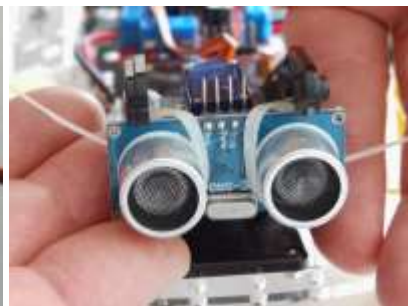
Now secure the base with a screw.



Attach the bracket to the chassis using the M2 screws delivered with the bracket.



From the bottom it should look like this.



Now attach the ultrasonic sensor with the pins up using two tie wraps as shown.



When done. Clip off the tie wrap ends with a wire cutter.

## 2-Battery Case and power wiring

Where applicable wires can be clipped to the right length, strip, twist and solder. It is not really needed but makes for a tidier result.



Stick two pieces of double adhesive tape to the 2-battery case and secure with a screw.



The case overlaps the servo bracket. The tape stops it from rotating



Solder the red wire from the 4-battery holder to the switch. (not the one you just attached)



Now solder the black wire from the 2-battery case to the switch.



Feed the black wire from the 4-battery case and the red wire from the 2-battery case through the hole to the top



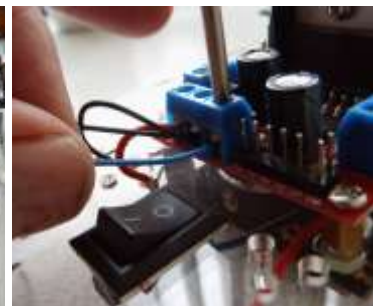
Connect the red wire to the left of the three terminal block.



The black wire from the 4-battery pack must be soldered to the separate black wire (ca. 15cm). Strip both wires. Twist the ends together and solder.

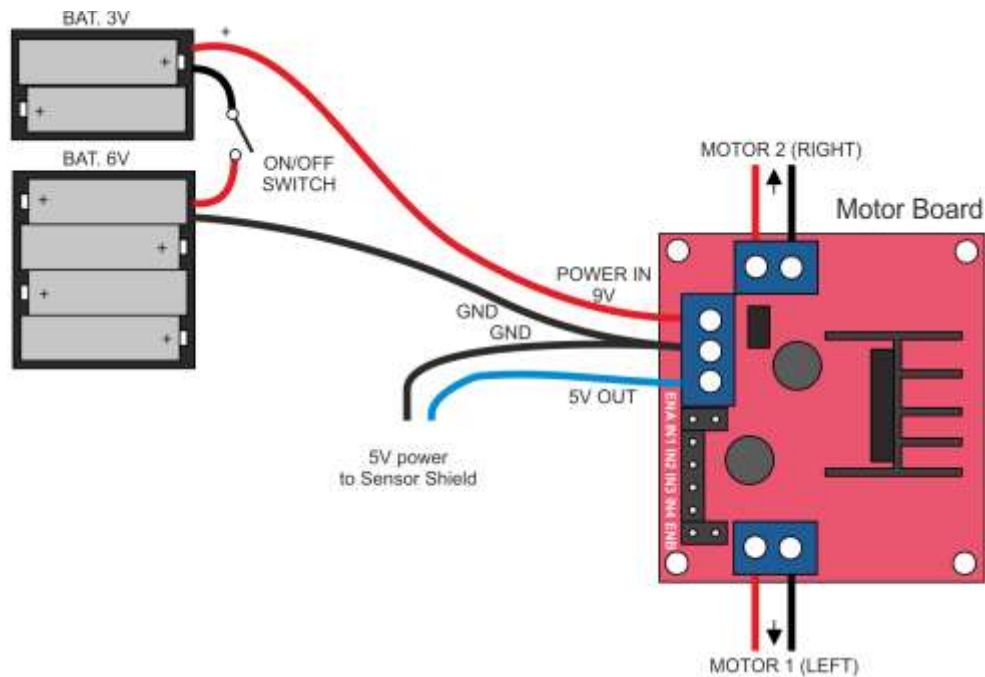


The two black wires connect to middle of the three terminal block. The separate black wire is the ground connection for the sensor shield.



The separate blue wire goes to the right most of the three terminal block. The blue wire provides the 5v power to the sensor shield.

In schematic



In the back of this manual is a complete schematic showing all connections.

The position of the switch is chosen to avoid unnecessary soldering. For the electronics it does not matter.

Be precise in these connections. A wrong polarity may damage the Arduino board.

In your kit another wire color might be chosen. Just see to it that the 5V out connects to VCC on the Sensor Shield / Arduino.



Feed the black and blue wire through the hole next to the standoff to the bottom.



And back up again next to the Arduino board underneath the terminal



The black wire goes to the left "GND" terminal, the blue wire to the right "VCC" terminal.

Now all the power wires are connected.

N.B. The 5V feed to the Sensor Shield and Arduino board is disputable. When connected to USB there are two power feeds working. To avoid more complex wiring we chose this somewhat ugly solution which however seems to work well enough.



## Motors

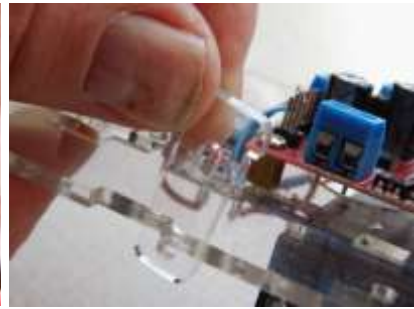
Now the motor wires need to be soldered.



Secure the motor wires with a tie wrap,



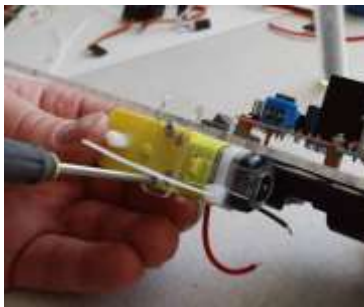
Check that the axels are firmly in place and attach the index wheel on the wire side of the motor



Place the first motor bracket in its slot.



Take the second motor bracket place the motor against it and put one of the long M3 screws through it. The wires should be on the inside.



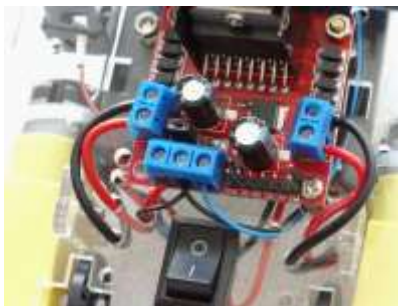
Fasten the screw with a nut on the inside and do the same with the second screw.  
Do not tighten too much!  
It will cause friction in the gears!



Do the same for the second motor.  
The result should look like this.



Now feed the motor wires through the hole in the chassis and attach them to the motor board.

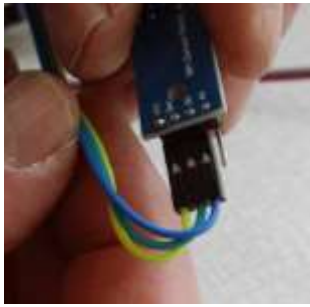


They can be clipped to the right length.  
The top wire from the motor goes to the rear (here black).

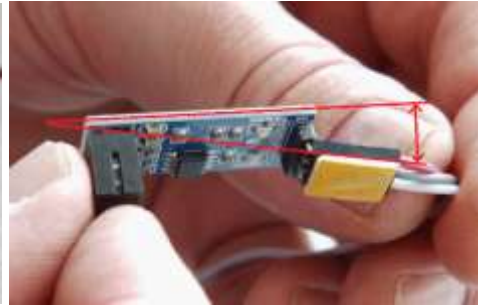


Attach the wheels.

## Speed Sensors



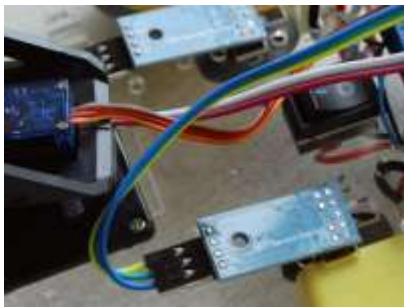
Take one of the speed sensors and attach a 3 wire cable to the VCC, GND and DO pins.



Bend the pins gently so they are under an angle like shown. Attach a piece of double adhesive tape underneath.



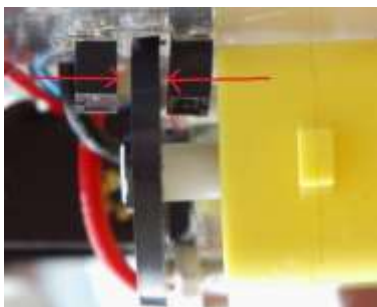
Now stick the sensor in its place using the tape. It should be centered in the chassis sensor slot above the index wheel.



Do the same for the other speed sensor.



Secure the end with a piece of insulation tape.



See that the index wheel can move freely between the sensor slot. You can move the index wheel slightly if needed.



Check that the wheels can move free from the motor screws. If not pull the wheel slightly outwards.

## Wiring

Now we have to connect everything together. In the back of this manual is a schematic showing all connections. It can help to understand what goes where. Remember that the cable colors are not important. They only help to tell the wires apart.



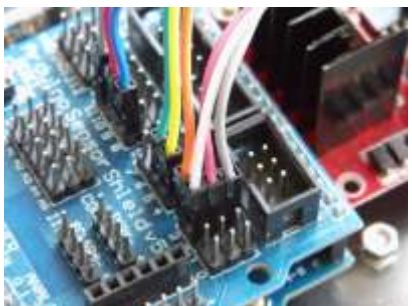
We start with the motor board. From this remove the two jumpers marked ENA and ENB.



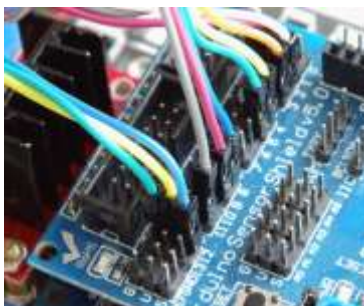
Attach the 6 wire Dupont cable to the pins ENA, IN1, IN2, IN3, IN4 and ENB in sequence on the motor board.



Attach the other side of these wires on the sensor shield to the pins on the S-row:  
ENA > 3, IN1 > 4, IN2 > 5  
IN3 > 8, IN4 > 9, ENB > 11.



Now connect the right speed sensor. This uses all pins (G,V,S) of lane 2.  
GND > 2G, VCC > 2V, DO > 2S.



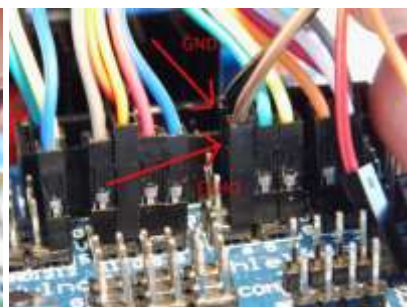
The right speed sensor goes to  
The G,V and S pins on lane 12.  
GND > 12G, VCC > 12V,  
DO > 12S.



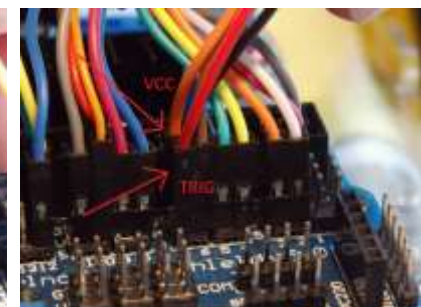
The servo wires are in one connector they go to lane 10.  
Brown > G10, Red > V10, Orange > S10.



Connect the 4 long wired (30 cm) dupont cable to the Ultrasonic sensor.



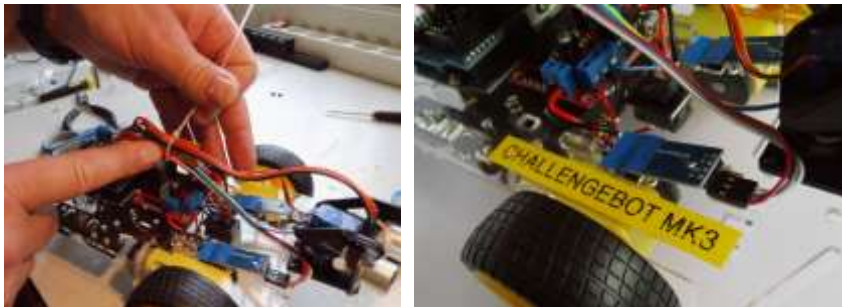
They connect to two lanes:  
GND and ECHO to lane 6.  
GND > 6G, ECHO > 6S



VCC and TRIG to lane 7.  
VCC > 7V, TRIG > 7S.



## Finishing off



Tie all leads together using the tie wrap. And put your (or a) name on it (So it won't get lost between all the other robots).

*Congratulations! You have just built a robot!*

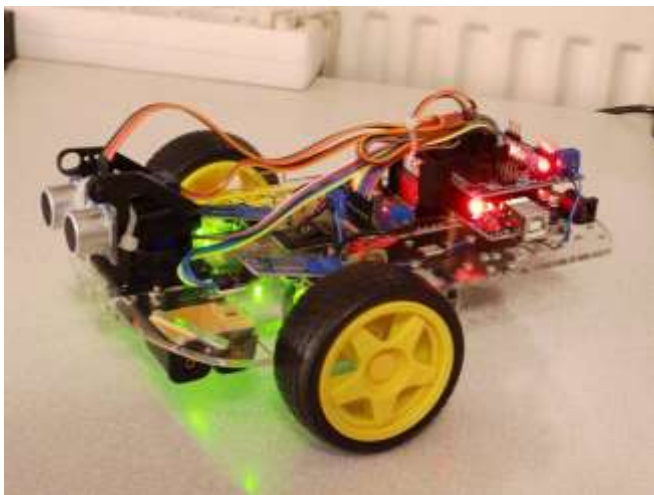
Now we need to bring it alive.

## Testing the hardware

### Power on test

Check the switch is in the off position.

Put in the batteries and switch on.



Several LED's should light up on the Arduino board, the Sensor Shield, motor board and the speed sensor boards.

Switch off again.

If no, or not all LED's came on, check the wiring and/or batteries.



## Installing the Arduino IDE

If you have not already done so, get the Arduino IDE from <https://www.arduino.cc/en/Main/Software>

Choose the windows or Mac OS X installer and install.

## Connecting the Arduino

Attach the USB-cable to the back of the Robot and your PC.

With a bit of luck your Arduino will be detected. In the Arduino IDE (under) tools you can choose “port” and select the port your Arduino is connected to.

In some cases you will need to find the right USB driver on internet.

## Get the robot examples


I have prepared two small programs RobotTestMK3 to check the hardware and RobotHelloWorldMK3 to have an example to start your own development. Both can be found at


<https://github.com/JaapDanielse/ChallengeBot.git>

## ChallengeBotTest

I have made a small program to test the hardware called RobotTestMK3.

Since the Arduino IDE requires a program source to be in a folder with the same name, you will find this in the folder “RobotTestMK3”.

Open the RobotTestMK3.ino file in the IDE and click on  to compile and upload to the Robot.

When done, open the monitor by clicking on . This will open a window in which you can see the robots output and send commands.

When testing the robot must be connected to USB and wheels must be free from the ground do they can turn without moving the robot.

The Arduino can be used via USB but the motors won’t do very well. If the battery is switched on all power wil come from the battery. For thet test you need the battery switched on.

Follow the RobotTest dialog and check if everything works all right.

You need a few values returned by the dialog to bring to the RobotHelloWorldMK3 sketch.

Write these down.

Servo 0° value:

Servo 90° value:

Servo 180° value:

Slow PWM:

Slow Speed:

## Developing your own software

To get you started and avoid endless experimenting in the limited time we have I made an example program “ChallengeBotHelloWorld” where the robot drives 1 meter, turns 180 deg. drives back again turns etc. The robot also stops if something is less than 5 cm in front of it and scans before the 180 turn. It is described later on.

I suggest to use this as a starting point to begin your own development.

## Welcome to the real world.

Our normal software development is fairly precise. A condition exists or it doesn't and you can make the software react to it. Developing the RobotHelloWorld I found things aren't as consistent as we should like.

For starters the motors are not built very precisely. So they differ in speed when given the same amount of energy. The wheels wobble a bit so the speed is not constant over the turn of a wheel.

I solved this (to some extent) by switching off or on a motor depending on the measured (and requested) speed and the position of one wheel compared to the other. But it does not always work right. Sometimes a wheel has turned more than expected or slipped and then we have changed direction without knowing.

The speed sensors work with an index wheel with 20 slots (and 20 spokes) the spokes however are smaller than the slots so to have a solid base for speed calculation we can only use the entering of a slot as an event. When stopping however we don't know how far away the next interrupt is. If this is earlier for one wheel than for the other the regulating mechanism will make a (unintended) slight turn.

I didn't solve that one yet. Have fun!

The ultrasonic sensor needs to be read swiftly during driving. But if you do it too quickly after each other. Then echo's still coming in from the previous measurement cause misreads.

I solved this by timing the distance between the requests and if too short repeat the previous answer. But still now and again some errors occur.

All these examples just to make you aware that in programming Robots things tend to be puzzling.

## Developing for Arduino

The Arduino's are based on a family of processors made by the Atmel company. The Arduino UNO has a Atmel ATMEGA328P processor which is a really amazing in capabilities, small and cheap processor. And perfect for this type of task.

The Arduino IDE lets you develop in C or C++.

Unfortunately you cannot debug from the Arduino IDE. It is however possible to use a plugin for MS Visual studio that has (limited) debug capabilities.

It is a good idea to keep in mind that variables are not initialized automatically.

## Controlling the Robot Hardware

### IO pins

From the software you can determine the use of the processors IO pins. They can be set to OUTPUT (to control something) or INPUT (to read a sensor). In the example you'll find this in the init routine of each module. A pin can be set to LOW or HIGH if OUTPUT and becomes LOW or HIGH when set to INPUT.

### PWM

Arduino also has a (more or less) analog output using PWM (pulse with modulation). This is a processor feature that switches a pin high and low, very fast, depending on the `analogWrite` value. We use this to control the energy going to the motors.

PWM is also used for the servo position. This requires a specific setting of the processors PWM logic.

### Motor Board

The motor board lets the motors work on a higher voltage and current than available from the processor and it controls the direction of the motors using H-bridge chips. The board can control 2 DC motors (the ones we're using).

The direction of the motors is controlled the IN(n) pins.

### Speed sensors

The speed sensors raise or drop a pin depending on light passing through a gate. A comparator chip on the sensor board makes the output clean.

### Interrupt

To make sure all events from the speed sensors are noticed we use an interrupt mechanism.

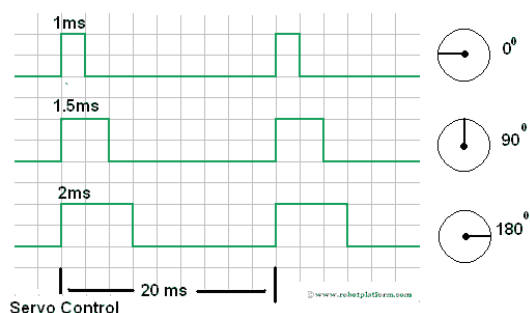
This reacts on the change of a pin from HIGH to LOW or reverse. The processor continues with the interrupt service routine. In `SpeedSensorInit()` and `ISR(PCINT1_vect)` You can see how this works.

### Ultrasonic distance sensor

The ultrasonic distance sensor has a trigger and an echo pin. If the *trigger pin* is raised for 10 us and then dropped again a ultrasonic sound burst is generated. From that moment we "listen" to the *echo pin* which becomes HIGH when the echo is "heard". Timing the return of the echo gives the means to calculate the distance.

### Servo

The servo can be positioned to a direction between 0 and 180 degrees. This is controlled through a pulse with a specific length. In our case this is done using PWM. It requires a specific setting of the processors PWM logic. Here a pulse of (about) 1ms means a 0 position and a pulse of 2 ms. a 180 position. The repeat frequency is 20 ms. This is all done by the PWM logic without CPU interference.



## ChallengeBotHelloWorld

Load the Arduino sketch form the folder “RobotHelloWorld” into the IDE.

This sketch consists of 4 modules:

### ChallengeBotHelloWorld

Main module this contains the standard setup() and loop() functions.

From this module the behavior of the robot is determined.

There are a few #defines at the beginning of the source. The SLOWPWM is likely to be dependent on the specific friction of the motors in your robot.

### Behavior pattern

The behavior is modelled in a switch statement where each case is an action and each action determines the next step or deviation.

### Drive

High level drive control. This module provides basic functions to drive in a straight line (more or less) and turn a desired number of degrees. It also scans for obstacles when driving and alerts the main module if something is too close. To do this it uses the low level routines in the other modules.

### Drive Functions

*boolean driveStraight(byte direction, int distance, int driveSpeed, byte obstacleDistWarning )*

drive in a straight line for a given distance with a given speed. Watch for obstacles closer then given distance

<i>byte direction</i>	Direction to drive FORWARD / REVERSE
<i>int distance</i>	Distance to drive in mm
<i>int driveSpeed</i>	Speed to drive with: SLOW / FAST
<i>byte obstacleDistWarning</i>	Distance in cm to warn for an obstacle
<i>boolean return</i>	returns false if obstacle encountered.

*void driveTurn(int turnDegrees)*

make a turn for a (approximate) given number of degrees

<i>int turnDegrees</i>	Number of degrees to turn. Positive is clockwise, negative counterclockwise.
------------------------	--

*void driveStop ()*

stop straight drive or turn by shortly reversing motors

*int driveDistanceDone()*

Returns the distance driven by the last driveStraight in mm.



## *MotorControl*

Low level routines to control the motors.

### *Motor control functions*

#### *void motorControlInit()*

Set up hardware for motorcontrol

#### *void motorControl(byte motorNumber, byte motorDirection, byte motorSpeed)*

Control motor direction and speed (sets the H-bridges)

<i>byte motorNumber</i>	Numer of the motor to control (1/2)
<i>byte motorDirection</i>	FORWARD / REVERSE / STOP
<i>byte motorPwm</i>	Pulse With Modulation value to output to motor

#### *void motorSpeed (byte motorNumber, byte motorPwm)*

Control motor speed without changing H-bridge setting

<i>byte motorNumber</i>	Numer of the motor to control (1/2)
<i>byte motorPwm</i>	Pulse With Modulation value to output to motor

## *SpeedSensor*

Low level controls to read the speed sensors. Uses interrupt mechanism which is extended to the Drive module via a callback routine.

### *SpeedSensor functions*

#### *void speedSensorInit()*

Initialize Speedsensor (setup hardware pins)

#### *void speedSensorSetDirection( byte sensorId, byte sensorDirection )*

Set direction for distance calculation

<i>byte sensorId</i>	Sensor number (motor nr) (1/2)
<i>byte sensorDirection</i>	FORWARD / REVERSE

#### *byte speedSensorGetDirection( byte sensorId )*

Returns the set direction for a given sensor (1/2)

<i>byte sensorId</i>	Sensor number (motor nr) (1/2)
----------------------	--------------------------------

#### *void speedSensorClear()*

Resets the speed sensor counters to 0

*int speedSensorReadCount( byte sensorId )*

Returns the interrupt count for a given sensor (1/2)

*byte sensorId*                      Sensor number (motor nr) (1/2)

*int speedSensorReadTime( byte sensorId )*

Return the time in ms since the last interrupt for a given sensor

*byte sensorId*    Sensor number (motor nr) (1/2)

### *DistanceSensor*

Reads the distance sensor and returns the measured distance in cm.

### *DistanceSensor functions*

*distanceSensorInit()*

Initialize Distance sensor (setup hardware pins)

*int distanceSensorRead()*

Returns measured distance in cm.

*boolean distanceSensorCheckObstacle(int obstacleDistWarning)*

Returns a true when an obstacle is closer than a given distance.

*int obstacleDistWarning*              Distance to an obstacle in cm

### *Servo*

Controls the servo and provides a sweep and sweep analysis function.

### *Servo functions*

*void ServoInit(int MinPulse, int MidPulse, int MaxPulse)*

Initializes the Servo function by setting up the hardware for the right type of PWM pulses on pin 10.

*int MinPulse*    Pulse length for 0 degrees position in ms.

*int MidPulse*    Pulse length for 90 degrees position in ms.

*int MaxPulse*    Pulse length for 180 degrees position in ms.

*void ServoWrite(int Degrees)*

Turns the servo in the given direction.

*int Degrees*      Number of degrees (between 0 and 180) you want the servo to turn to.

*void ServoSweep(int SweepArray[])*

Makes a sweep starting at 90 deg. Turning backwards to 0 deg. forward to 180 and then back to 90 deg. Providing 60 measurements (one per 3 degrees) in an interlaced pattern.

*int SweepArray[]*            The sweep array

*boolean SweepAnalyse*

*( int SweepArray[], int MaxDistance, int MaxWidth, int \*Direction, int \*Distance)*

*Analyses the sweep array and returns the direction and distance to closest fitting object.*

*int SweepArray[]*            The sweep array

*int MaxDistance*            Maximum distance an object can be away to be noted

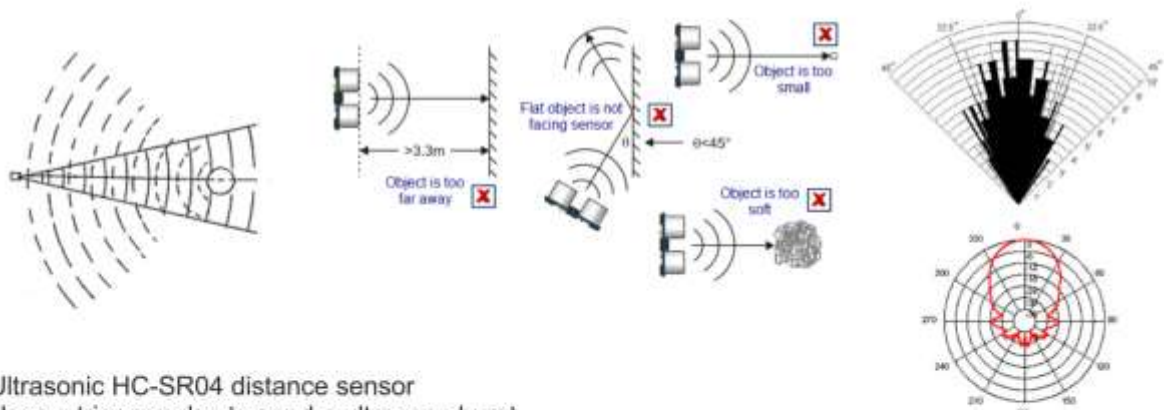
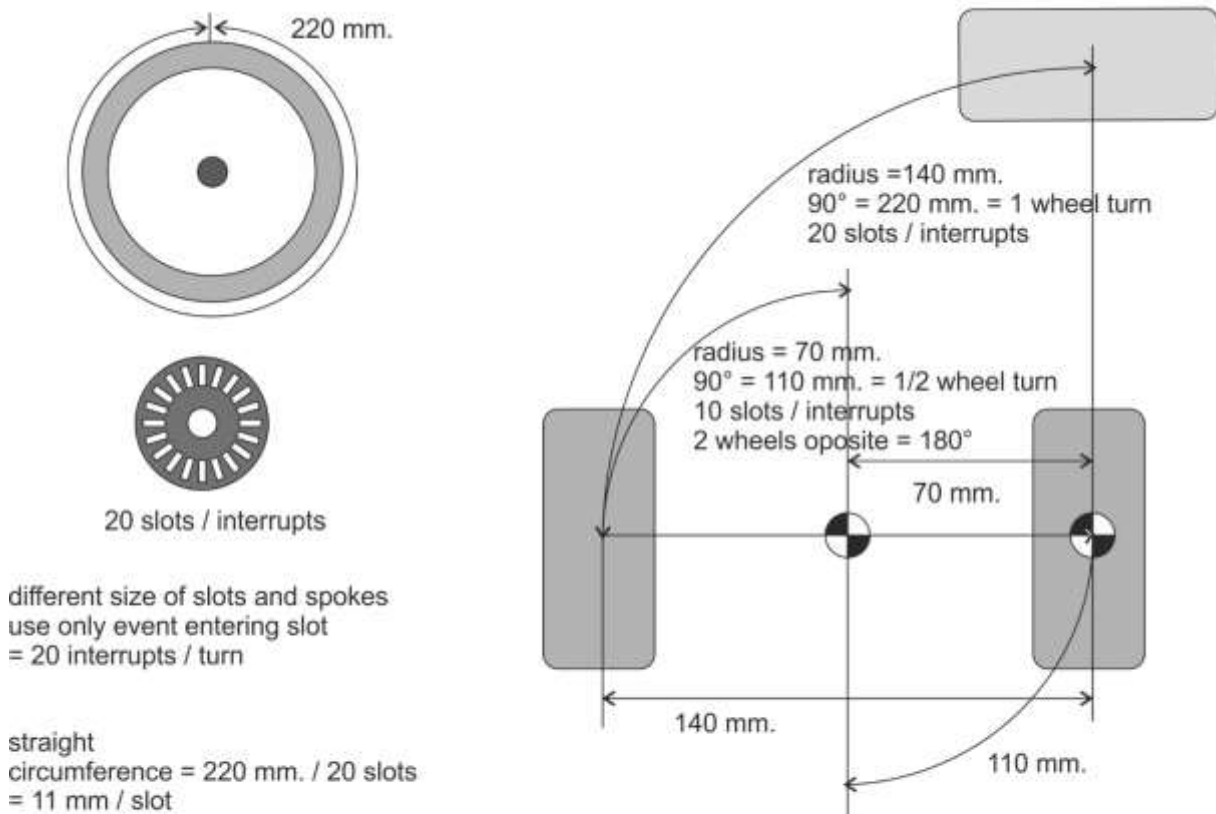
*int MaxWidth*                Maximum number of measurements an object can be to be noted

*int \*Direction*              Return address for the direction of the object.

*int \*Distance*                Return address for the distance to the object.

## Calculations

sizes and calculations



Ultrasonic HC-SR04 distance sensor

Uses a trigger pulse to send a ultrasonic burst.

Then waits for the echo to return

The duration in microseconds divided by 2 (to object an echo back)

and the result divided by 29.14 gives the distance in cm.

(29.14 = time in microseconds it takes sounds to travel a distance of 1cm)

After the measurement we need some time for echo's to decay.

If a next measurement is requested within 30 ms. the previous value is returned

\* This applies to the MK1 and 2 kit. In the MK3 kit the wheel circumference is 210 mm. This means the calculations are a factor 1,05 off. (Let's ignore this. This makes calculations much easier).



## The Challenge

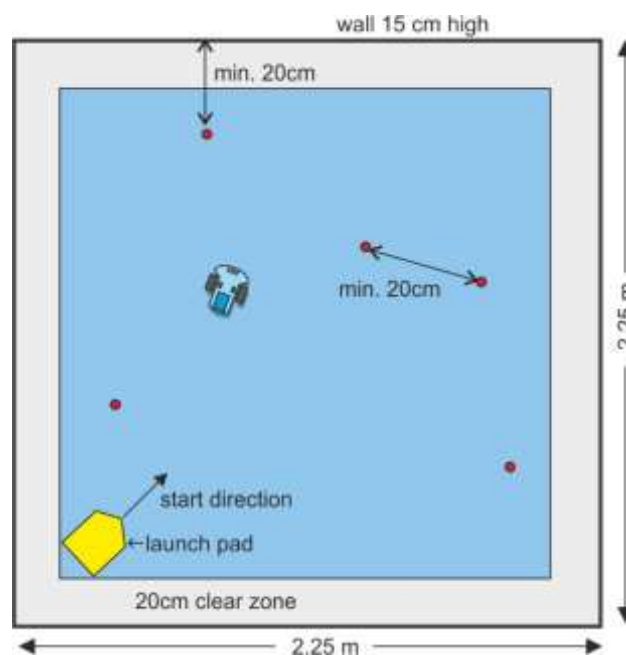
When you have built your robot you can make a sniffer bot that just drives around.

But here is a challenge you can do in a competition:

The Challenge is to knock over 5 targets within an area surrounded by “walls” as quickly as possible without hitting the walls. You get points for knocking over the targets but lose points for hitting the walls. There is a time limit but time left after having knocked over all targets grants you extra points.

### Arena

- The arena will be square 2.25 x 2.25 m. with 15 cm high walls.
- There will be 5 targets 12.5 cm high, 1.6 cm across.
- Targets are placed at least 20 cm from the edges and at least 20 cm from each other.
- The launch pad will be in one corner 20 cm from the sides facing the opposite corner.
- There will be a time limit of 3 minutes.
- If all targets are knocked over time will stop.
- An arena will have a referee. Robot modifications need to be approved by the referee.



### Points

- Knocking over a target 10 points
- Touching the wall -1 point.
- At the start a robot has 30 points for time.
- For every 6 seconds 1 point is subtracted.
- If all targets are knocked over the remaining time points will be added to the score.
- If a robot gets stuck against the walls it may be restarted at the launch pad. This will cost 5 points. Time will continue.

## Robot Schematic

