## **Current Issues:**

* We will need some permissions on DICE. Calum is going to try this out this week
* We need a micro SD Card (minimum 2GB, max 16GB) per robot plus a few adapters to either USB or micro-sd-to-regular-sd.
  + x30 16GB Micro SDHC from Kington: £120 on Amazon
  + Need to set up all 25 SD cards.
* Update tutorial scripts on IVR website. Remove matlab-robotics material

## **Setup: Installing ev3dev on SD Card**

Setup time was about 40 mins is downloading and installing ev3dev on an SD card

1. Install Ev3Dev on SD card (plugged into your computer). Then boot the ev3 with SD card installed:

<http://www.ev3dev.org/docs/getting-started/>

* Set up connection to the internet with “Ethernet over USB”. When its functioning you should be able to “ping google.com”
* restart the ev3 after doing all these steps as sensors might not work until after the first reboot. (so don’t do the “do something awesome” step)
* username: robot /// password: maker

1. Install the python bindings for the ev3dev: <https://github.com/rhempel/ev3dev-lang-python#python2x-and-python3x-compatibility>

## **Typical Usage**

### **Starting Up:**

1. Plug in USB into computer and brick
2. Press square button on ev3. boots through the ev3dev penguin screen and the brickman screen to (showing file browser, device browser etc)
3. On computer, select ev3dev network interface (created during the install)
4. Ssh onto the computer: ssh robot@ev3dev.local
5. You are now connected to the robot
6. (When finished shut down by pressing the upper right button a few times)

### **Development Cycle:**

You will develop your code in a text editor of your choice (in python) on your computer. When you want to test it on the robot, you will copy your entire code directory to the robot. This way you can keep your latest files on your computer at all times:

1. Write code on your computer.
   1. Make sure your script is executable: chmod +x test.py
2. Copy the code onto the ev3 to test: **scp -r ivr\_directory ev3:/home/robot/**

(follow the instructions below to set things up)

1. You can then execute the python script from the ev3 block UI by selecting it through ‘file browser’. Unplug the USB cable and press the square button on the EV3
   1. You can also execute it directly from a terminal over usb to avoid having to reconnect or to easily see terminal printing
   2. Make sure to make
2. It’s ALIVE! The robot will operate autonomously
   1. To quit a program press the upper right button for about 3 seconds

## **Robot Drill Lab 1:**

Basic Operation:

1. Build the simple wheeled vehicle: one motor, two wheels. Two touch sensors. Make sure to connect the switches to numbers and motors to letters, as shown
2. Connect to the robot as described in ‘Typical Usage’
3. Progressively work through the tutorials in main.py. You can run main.py from either a terminal or through the ev3’s navigation UI.
   1. Open Loop driving
   2. Turning an LED on and off with a switch
   3. Driving backwards and forwards using one switch to command each direction
   4. Building your first class to keep some state of robot

play with servo speed and led. use ultrasound sensor to activate the wheels

get to the stage that can use one touch sensors to drive forward and other to drive back. and operate leds

Build reference robot with color sensor facing downward and a gyro on top (and the wheel symbol facing up)

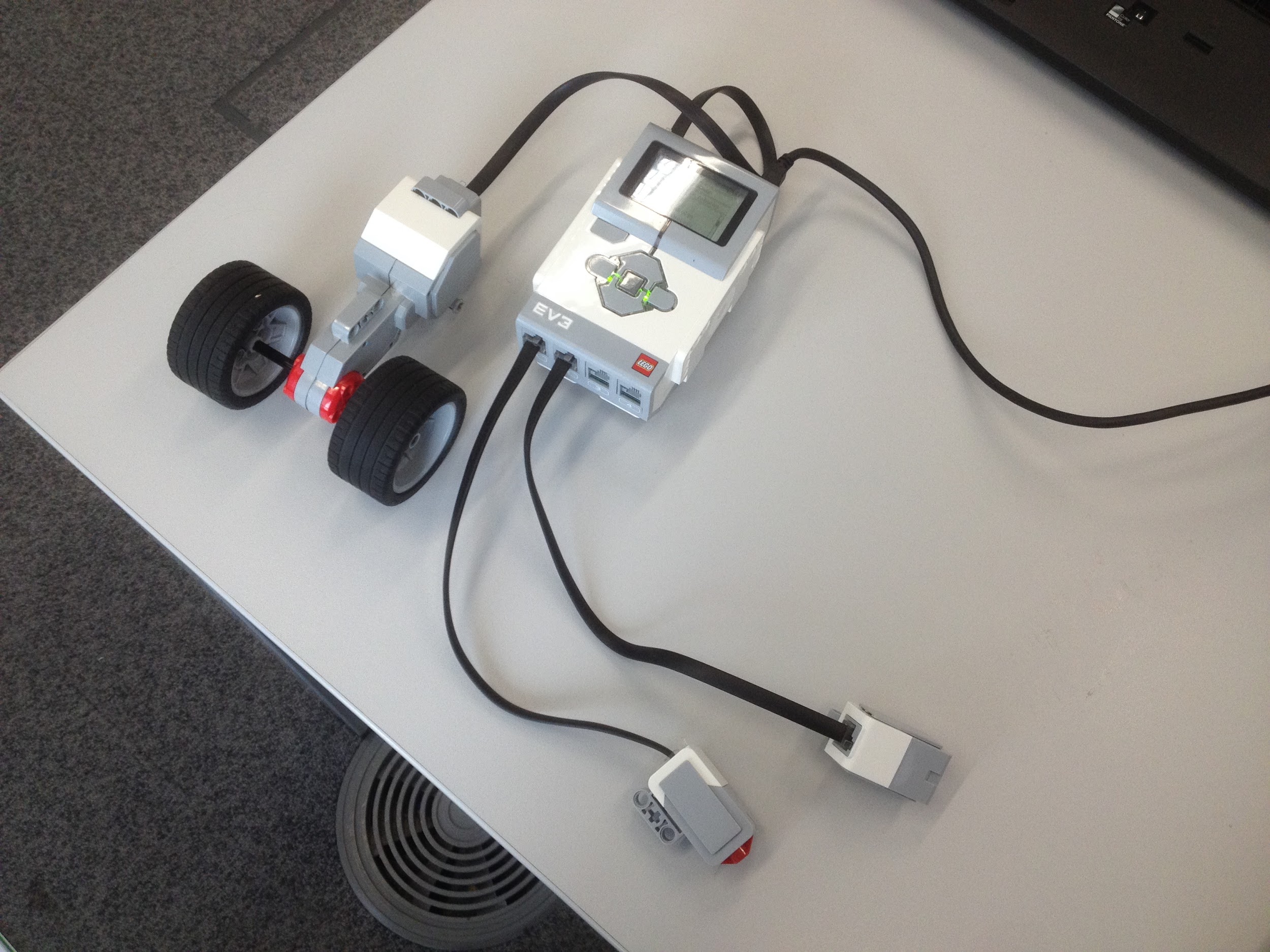


Fig: A Very Basic Robot - just wheels, a motor and two switches

## **Robot Drill Lab 2:**

A. Obstacle position estimator

1. Attached the MediumMotor and the sonar servo to the robot.
2. Build a detector to servo around and find the obstacle in front of the sensors
3. Explore what the servo can and can’t see e.g. how wide is the beam that it detects
4. Determine the range from the IR reading

Snippet of code to print the ultrasonic sensor reading. It returns millimetres:

us = ev3.UltrasonicSensor()

while True:

print us.value()

**IMPORTANT:** make sure that no other objects are in range of the sensor when testing. Otherwise it will get confused.

B. Wheel Odometry (plus gyro):

1. Develop a wheel odometry module to allow the robot to calculate how far it has gone:
   1. for the following commands, determine where the robot ended up and what the motor encoder and gyro readings were before and after?
      1. 2 seconds forward at 25%
      2. 4 seconds forward at 50%
      3. 2 second turn at 25% ...
   2. Repeat the experiment for one of the tests. How repeatable is each run? Was it better to start slowly and slow down gradually? How would it change if you added more weight to the robot or moved the wheels?
2. Develop a class to command the robot to go to a relative goal. open loop. This will use what you learned above.
   1. Given an goal relative to where the robot is - [x,y, yaw]:
      1. Rotate towards the goal (turn until the gyro reads the desired heading)
      2. Drive to the goal and stop when the encoder reading matches what you would expect.
      3. Turn to face in the required final yaw (correcting for the first turn)
   2. How repeatable is this?

Snippet of code to print the gyro reading. It returns degrees:

g = ev3.GyroSensor()

while True:

print g.value()

Snippet of code to operate the motor and read the encoder

m = ev3.LargeMotor('outA')

m.connected

print m.position

m.run\_timed(time\_sp=3000, duty\_cycle\_sp=75)

print m.position

Outcomes:

* Teaches what the sensor can see
* Develops obstacle detector for assignment
* Learn about gyros and odometry
* Build navigation module for assignment

## **Robotics Assignment**

**Part 1: Line following**

Develop an algorithm to follow a black line on top of a white piece of paper. Robot starts wherever the group would like to place it

**Part 2: Broken line segment following**

Follow a series of 4 line segments in a left-right pattern as shown. Robot will start on a left hand line and drive along it before switching to the other line.

* *Have robot speak the state when switching: "have reached the end of the line and will search on the right for the next line”*

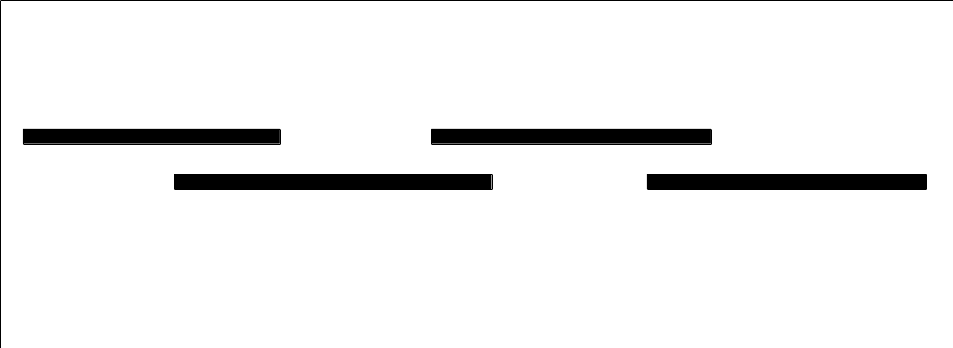


Fig: Broken Line Segment Course. Starts on left sided line

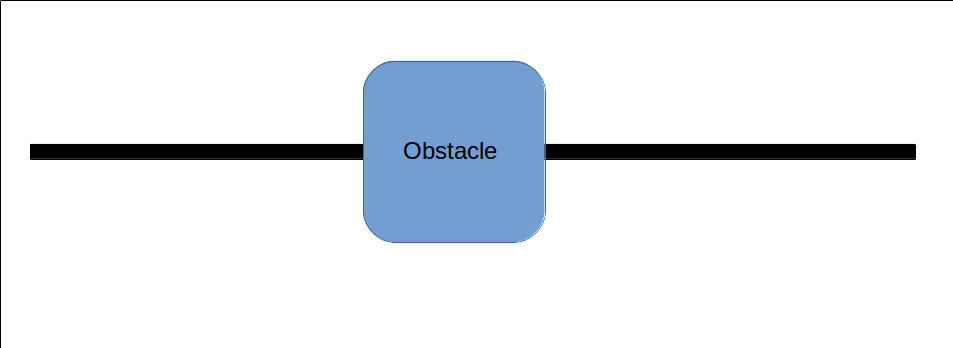
**Part 3: Build a servoed sonar obstacle detector**

Have the sensor servo over and back. When it sees something find out the closest range by servoing more slowly.

* *Have the robot speak when it’s found the obstacle and how far away it is.*

**Part 4: Follow a line to a obstacle. circumvent the obstacle, pick up the line again follow to the end**

Use the ultasound sensor to avoid driving in the obstacle. Keep the obstacle at a safe range when driving around it. Don’t need to use odometry as line will be present



**Fig: Obstacle blocking line segment**

**Part 5: Follow a line to a obstacle. circumvent the obstacle, then drive to a goal**

There is no line to detect on the otherside of the obstacle. use odometry to make it to the opposite side of the obstacle and face towards the perpendicular line

* *acoustically declare "I am pointing at the line" and then drive to and stop at the line*

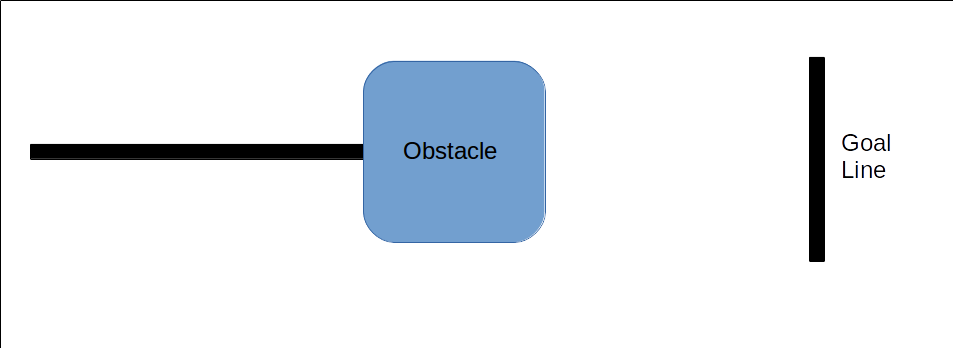


Fig: We will use different sized blocks so that odometry will be necessary

*NOTES FOR MARKING:*

*Part 2: marker will put a wooden block on left, right, front of robot and the sensor will servo around until it finds it.*

*Part 3 & 4: Marks taken off for driving into obstacle*

## **Setting up a Development Environment**

To do this we suggest setting up password-less login and ssh aliases. After that it will be just one command to send all contents to the robot

### **SSH Login without needing to enter a password**

Purpose: be able to log into ev3 as ‘robot’ from YourComputer without entering the password each time. From: <http://www.linuxproblem.org/art_9.html>

1. First log in to your computer and generate a pair of authentication keys. Do not enter a passphrase:

yourname@YourComputer:~> ssh-keygen -t rsa

Generating public/private rsa key pair.

Enter file in which to save the key (/home/yourname/.ssh/id\_rsa):

Created directory '/home/yourname/.ssh'.

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /home/yourname/.ssh/id\_rsa.

Your public key has been saved in /home/yourname/.ssh/id\_rsa.pub.

The key fingerprint is:

3e:4f:05:79:3a:9f:96:7c:3b:ad:e9:58:37:bc:37:e4 yourname@YourComputer

2. Now use ssh to create a directory ~/.ssh as robot on ev3dev.local (The directory may already exist, which is fine):

yourname@YourComputer:~> ssh robot@ev3dev.local mkdir -p .ssh

robot@ev3dev.local's password:

3. Finally append your new public key to robot@ev3dev:.ssh/authorized\_keys and enter robot’s password one last time:

yourname@YourComputer:~> cat .ssh/id\_rsa.pub | ssh robot@ev3dev.local 'cat >> .ssh/authorized\_keys'

robot@ev3dev.local's password:

4. You should now be able to ssh to ev3 without using a password

yourname@YourComputer:~> ssh robot@ev3dev.local

### **Creating short SSH aliases**

Purpose: To be able to make a short alias of the ssh script:

1. On your computer, add these contents to .ssh/config:

Host ev3

User robot

HostName ev3dev.local

### **Final Result**

* To ssh onto the robot: **ssh ev3**
* To scp a file onto the robot: **scp test.py ev3:/home/robot/**
* To scp an entire directory onto the robot: **scp -r ivr\_directory ev3:/home/robot/**

## **Older Stuff**

### **About EV3DEV**

* Homebrew opensource access to the EV3
* Supports C++ and Python. (Possibly ROS).
* Calum could access it via a wifi dongle but had problems with certain wifi dongles and using more than one dongle
* There exists some demo programs using IR sensors but we only have about 4 IR sensors. They cost £26 from Lego. (Might have Issues buying in bulk)
  + We have ultra sound sensors but they are of poor quality

### **Assignment in 2015/16**

Made a 2 wheel balancing robot.

* Assignment 1: Used code found online to carry out PID control for balance. focused on parameter tuning of PID by students: modifying dials in a lego GUI
* Assignment 2: Extended this for one of these navigation tasks. This requires some programming using as sandbox application students found online
  + Line following (half of class did this). This was successful
  + Obstacle avoidance (half of class did this)
  + Driving in a square

Issues:

* Required window on the student’s laptops. We cant assume this
* Lego GUI becomes laggy when more than 12 blocks were added