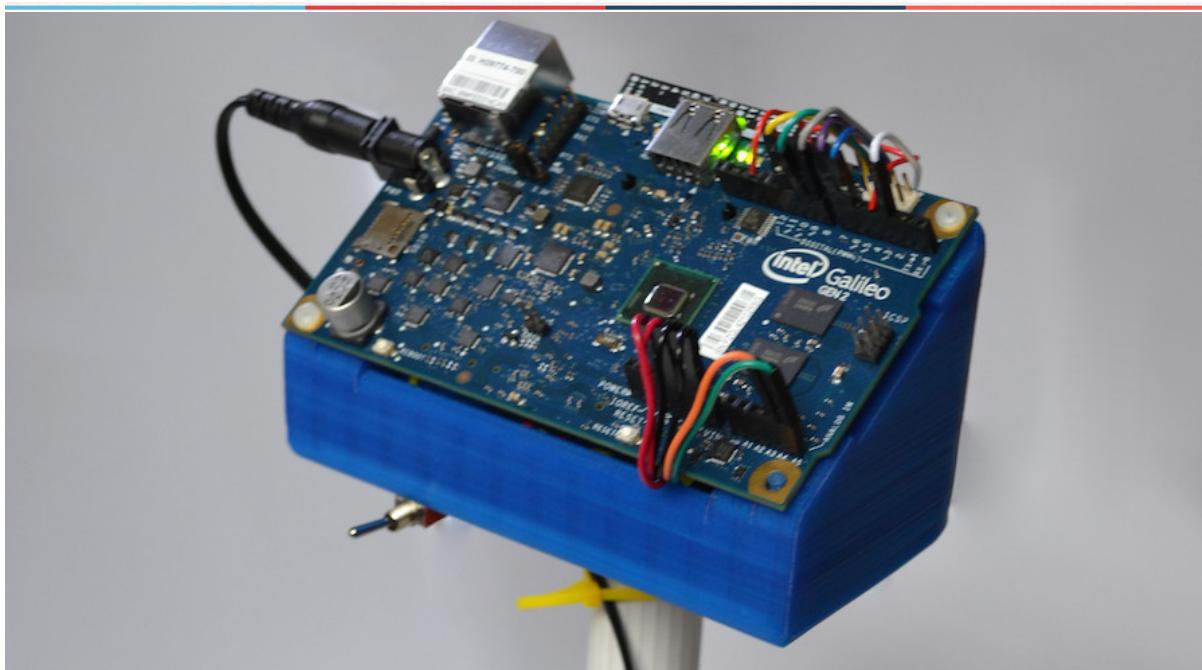
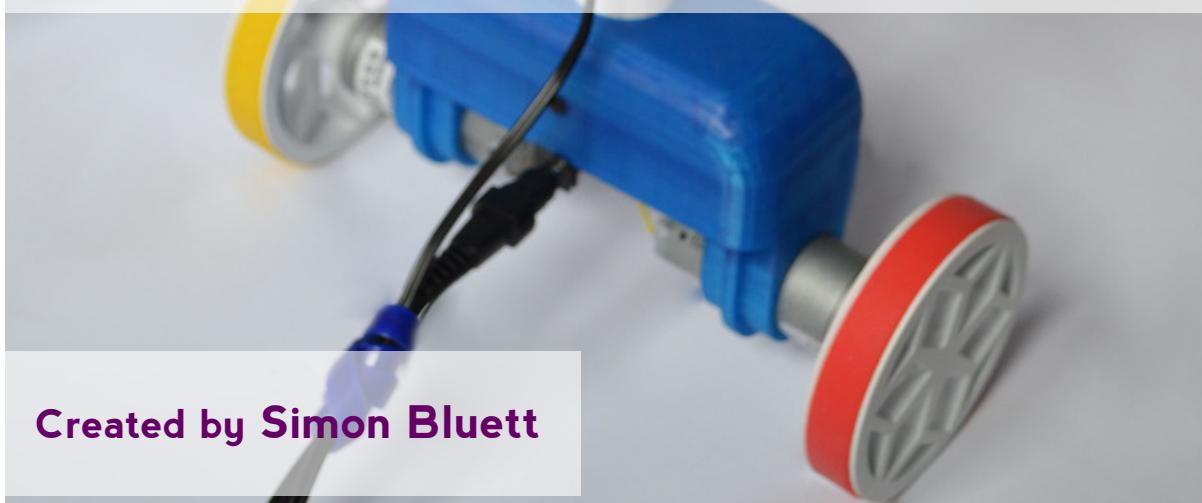


BALANCING ROBOT



DEMONSTRATION
Intel Galileo Development Board



Created by Simon Bluett

CONTENTS

	Page
1. Introduction	3
1.1 – Intel Galileo Development Board Gen 1 / Gen 2	3
1.2 – Balancing Robot Demonstration	3
2. Components and Requirements	4
2.1 – Hardware Requirements	4
2.2 – Software Requirements	4
3. Assembly Instructions	5
3.1 – Full Assembly	5
4. Programming Instructions	9
4.1 – Computer Software	9
4.2 – Installing Libraries	9
4.3 – Uploading the Balancing Program	9
4.4 – Optimizing the Performance	10
5. Demonstration and Usage	11
5.1 – Pre-demonstration Checklist	11
5.2 – Demonstration Procedure	11
5.3 – Post-demonstration Checklist	11
6. Troubleshooting	12
6.1 – Common Problems	12
6.2 – Other Issues	13
7. Conclusion	14

INTRODUCTION

1.1 Intel Galileo Development Board Gen 1 / Gen 2

Intel Galileo is the first in a line of Arduino-certified development boards based on Intel x86 architecture and is designed for the maker and education communities.

Intel Galileo combines Intel technology with support for Arduino ready-made hardware expansion cards and the Arduino software development environment and libraries.

The development board runs an open source Linux operating system with the Arduino software libraries, enabling re-use of existing software, called "sketches". Intel Galileo can be programmed through OS X, Microsoft Windows and Linux host operating software. The board is also designed to be hardware and software compatible with the Arduino shield ecosystem.

Intel Galileo features the Intel Quark SoC X1000, the first product from the Intel Quark technology family of low-power, small-core products. Intel Quark represents Intel's attempt to compete within markets such as the Internet of Things and wearable computing. Designed in Ireland, the Quark SoC X1000 is a 32-bit, single core, single-thread, Pentium (P54C/i586) instruction set architecture (ISA)-compatible CPU, operating at speeds up to 400 MHz.

1.2 Self-balancing Robot Demonstration

This demonstration uses the concept of an inverse-pendulum system in order to stabilize itself with an accelerometer-gyroscope module and two motors. As the unstable system starts to tip over in one direction, the accelerometer-gyroscope records this change in angle, as well as the angular velocity. In turn a PID (Proportional, Integral, Derivative) controller is used to control the motors, therefore stabilizing the system.

The aim of this demonstration is to visualise the versatility and processing power of the Intel Quark and Galileo Development Board Series. Filtering and processing the data from the accelerometer-gyroscope module in real-time requires quite a large amount of processing power. Therefore the quick response rate of the Galileo Board – especially Gen 2 – will be highlighted in this demonstration.

Finally, this demonstration is visually appealing and entertaining and therefore is an excellent project which can be used to impress your friends and colleagues! This document describes only the setup of a basic self-balancing robot. Feel free to expand to this project and develop your own variant of this robot. If you do, please let us know about your progress at: <https://communities.intel.com/community/makers>

Or on my personal blog:

<http://wired.chillibasket.com>

What will you make? ☺

COMPONENTS & REQUIREMENTS

2.1 Hardware Requirements

The Galileo Self-balancing Robot demonstration requires the following physical components. Some of the parts are custom designed, and therefore have to be 3D printed. The *.stl 3D CAD files should be supplied along with this document. (If you do not have access to a 3D printer, you can design your frame from any parts you have lying around!)

Electronic Components:

- 1 x Intel Galileo Development Board (preferably GEN 2)
- 2 x 12v Power Supply
- 1 x [Official Arduino Motor Shield R3](#)
- 1 x [MPU-6050 Accelerometer-Gyroscope Breakout Board](#)
- 2 x [High Torque Gearbox Motors \(~12v, between 100 – 300 RPM\)](#)
- 1 x [External Power Jack](#)
- 1 x [SPST Toggle Switch](#)
- 1 x [PTM Button](#)
- Colour-coded wires (~1.5m), preferably single core

Mechanical Components:

- 1 x Top Component (Custom 3D design)
- 1 x Central Component (Custom 3D design)
- 1 x Base Component (Custom 3D design)
- 2 x Wheel Component (Custom 3D design, [designed by: CodeCreations](#))
- 4 x Nylon* Bolts, ø 3mm (Max 25mm long)
- 4 x Nylon* Nuts, ø 3mm
- 4 x Steel* Bolts, ø 5mm (Max 20mm long)
- 4 x Washers, hole ø 5mm
- 4 x Steel* Nuts, ø 5mm

* The material doesn't really matter; these are just the components I used when I made this project.

Other Physical Requirements:

- Soldering Iron with fine solder
- Wire strippers / Pliers
- Small Screwdriver
- Super-glue / All-purpose Glue
- Electrical Insulation Tape

2.2 Software Requirements

The Self-Balancing Robot demonstration currently uses the Arduino IDE interface in order to upload the required program onto the Galileo Board. (An improved version is currently being developed, which will bypass the Arduino interface and require you to connect directly to the Linux system.) Therefore the requirements are as follows:

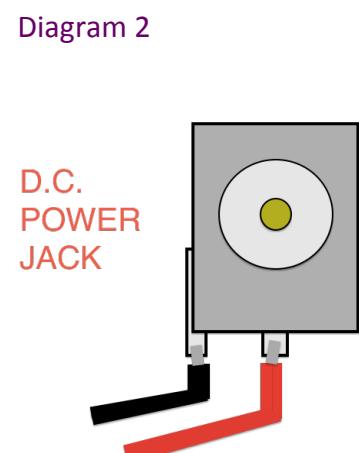
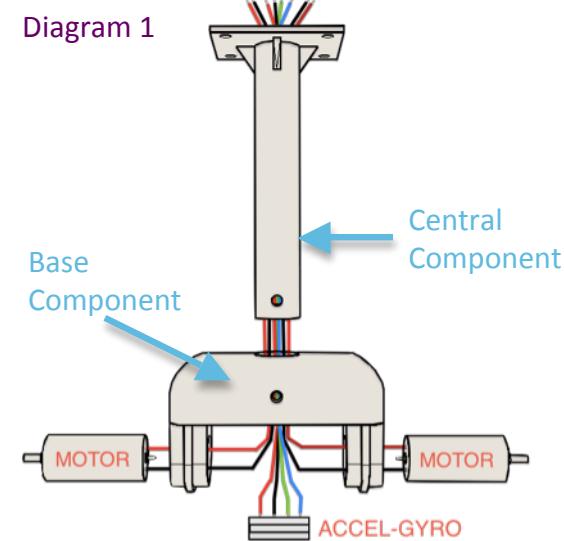
- Latest Version of the Galileo Arduino IDE (See [Programming Instructions](#))
- Compatible drivers for the Galileo Board (See [Programming Instructions](#))
- Self-Balancing Demonstration Code (As provided with this document)

ASSEMBLY INSTRUCTIONS

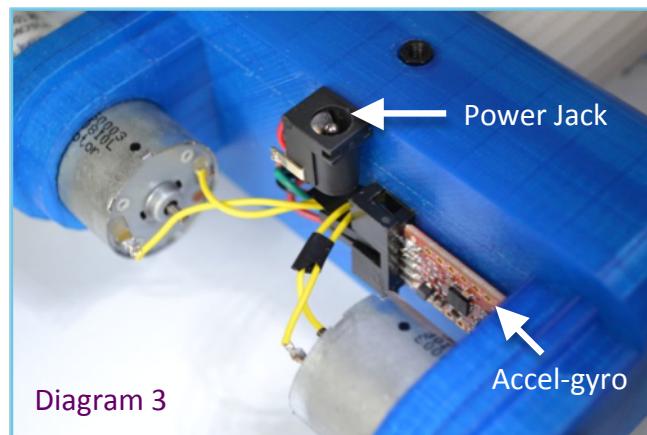
The structural assembly of the Self-balancing Robot demonstration is extremely simple as the main components were designed to slot into each other easily. The tricky part is getting all of the wiring correct. These assembly instructions aim to make the process as simple and as straightforward as possible.

3.1 Full Assembly

1. Before starting into the full assembly, test all of the electronic components to make sure that they are working properly.
2. Solder power cables onto both of the **motors**. The cables should be long enough to reach from the motors at the bottom of the robot, to the motor controller situated at the top, with a small amount of slack. ([Diagram 1](#))
3. Solder/attach wires to the Accelerometer-Gyroscope breakout module. Use different coloured wires for **3.3v**, **GND**, **SDA** and **SCL**. The interrupt pin will not be used, as the Galileo cannot deal with interrupts yet. ([Diagram 1](#))
4. Solder two wires onto the **Power Plug**. ([Diagram 2](#)) Once again, the wires should be coloured for clarity, and long enough to reach the top of the robot.

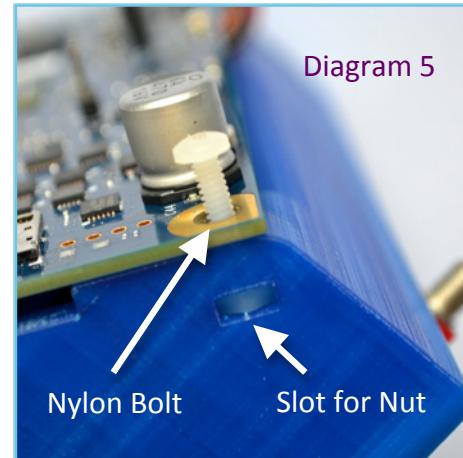
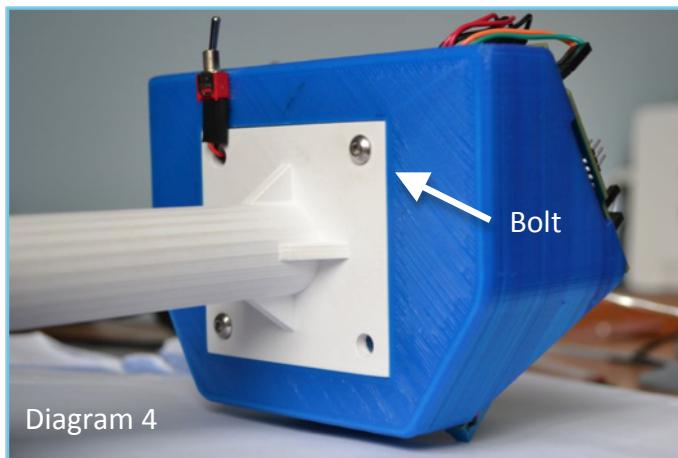


5. Slot the motors, the accelerometer-gyroscope module and the power plug into their respective slots on the **base component**. Use a small drop of glue on the accelerometer-gyro and the power plug in order to keep them firmly in place. ([Diagram 3](#)) Make sure all of the wires are drawn fully up through the centre of the **Central Component**.



6. Attach the plate of the **Central Component** into its slot on the bottom of the **Top Component**. The **Top Component** should be positioned with the Galileo-side facing in the opposite direction of the power plug.
7. Pull all of the wires through the provided hole in the **Top Component**.
8. Using the four metal bolts, washers and nuts, fasten the **Central Component** firmly onto the **Top Component** using the four holes provided. (Diagram 4)

9. Fasten the **Central Component** to the **Base Component** by sliding a bolt into the provided hole in the base.
10. Slot the **nylon nuts** into their respective place in the **Top Component** as shown. (Diagram 5)



11. Complete all of the wiring as shown in the diagrams below. The diagram shows a Galileo GEN 1 board, instead of a GEN 2 one, but the wiring is still the same. (Diagram 6 & 7 – On pages 7 & 8)
12. Glue the push-button switch into the hole on the side of the **Top Component**, and glue the toggle switch onto the underside of the **Top Component**.
13. Fasten the **Motor Shield R3** and the **Galileo Board** into their proper positions using the provided notches and two nylon bolts per board.

14. Insert the motor wires and the external power wires into the screw terminals on the Motor Shield. The left motor connects into the **A-motor Terminal**, and the right motor connects into the **B-motor Terminal** (The Galileo Board faces forwards). You won't really know which wire goes into the positive and which goes into the negative terminal until you have tried it out! (Diagram 8)
15. Ensure that your assembly matches the image of the completed robot depicted below! (Diagram 9 & 10 – On page 8)

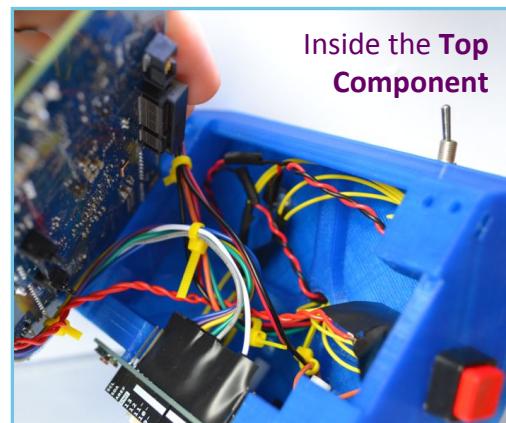


Diagram 6 - Schematics

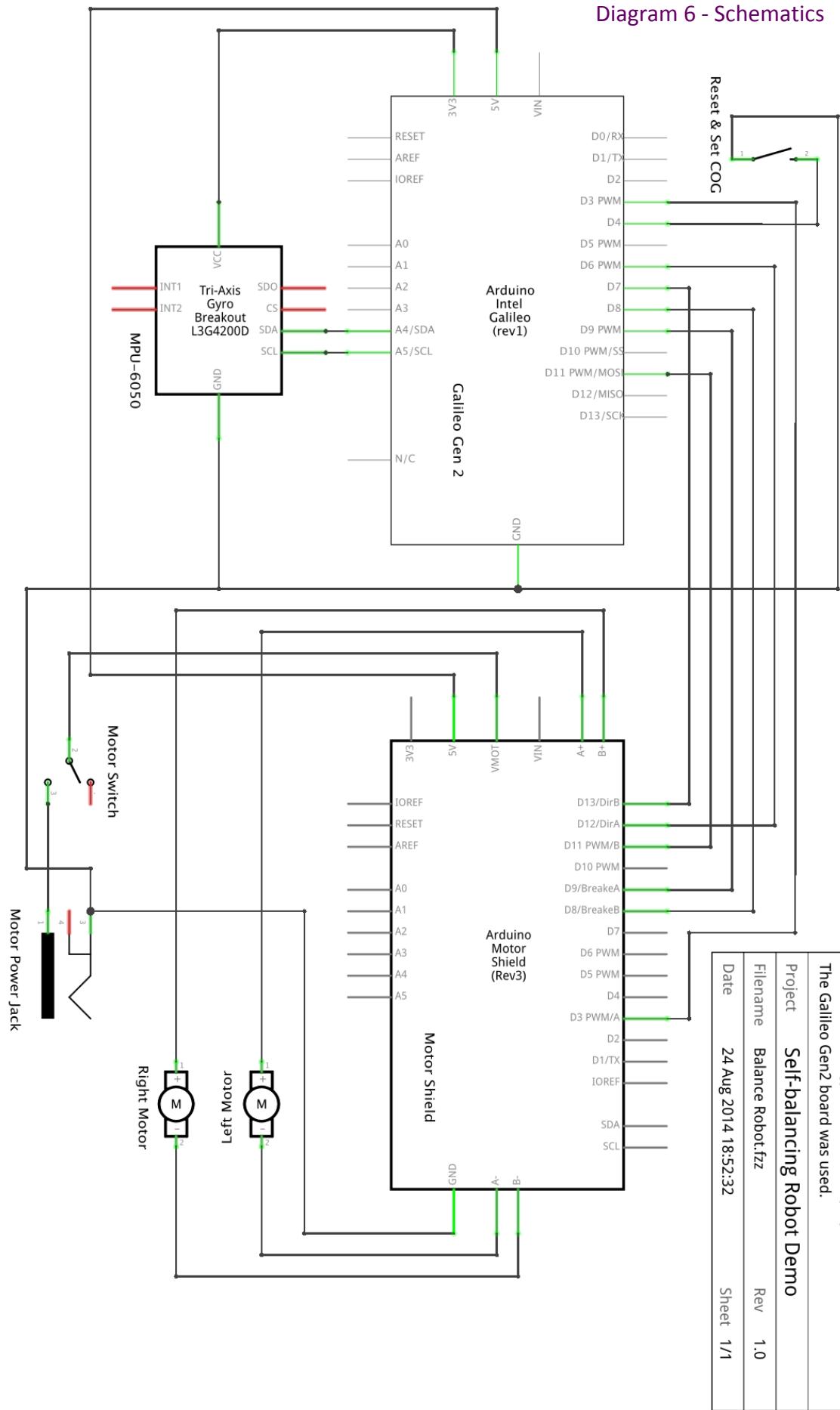
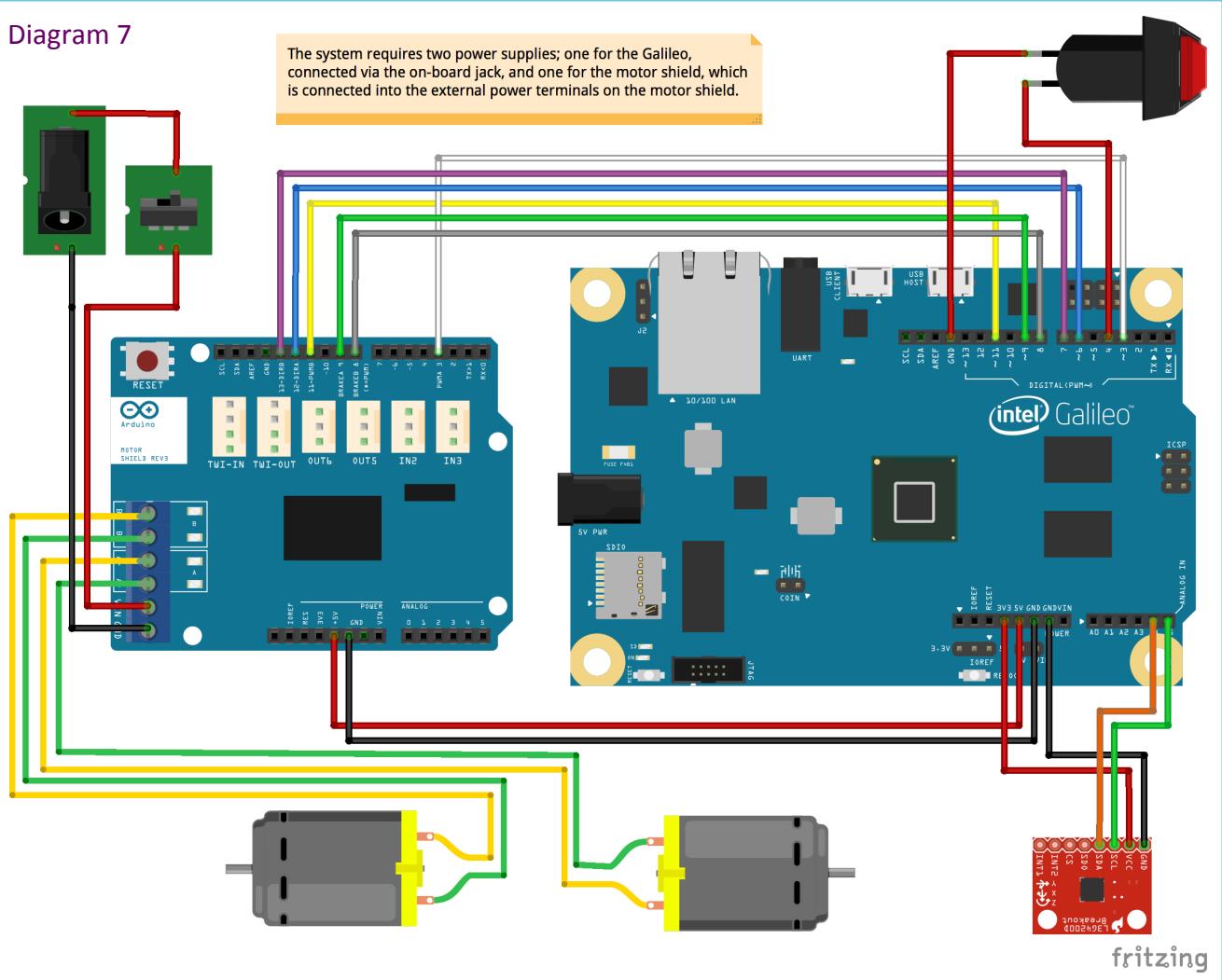


Diagram 7

The system requires two power supplies; one for the Galileo, connected via the on-board jack, and one for the motor shield, which is connected into the external power terminals on the motor shield.



fritzing

Diagram 9

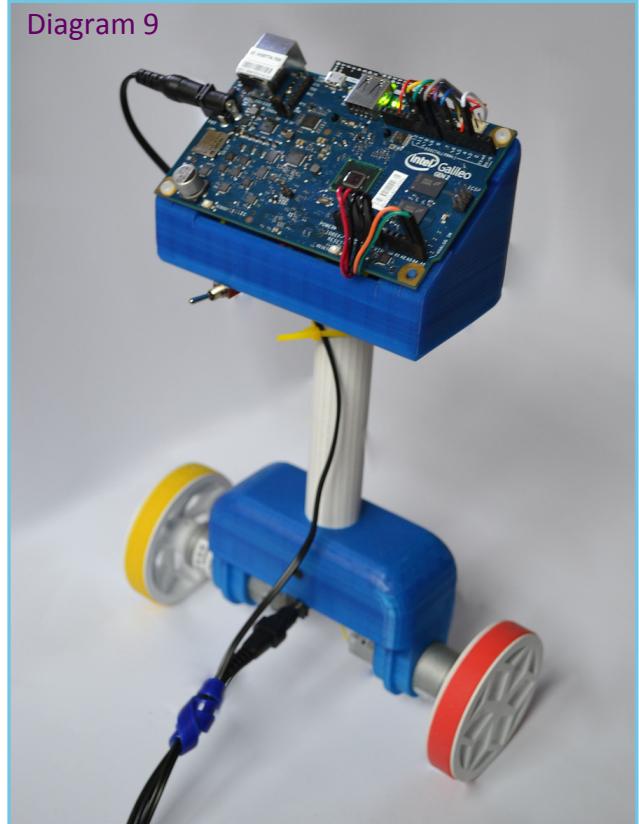
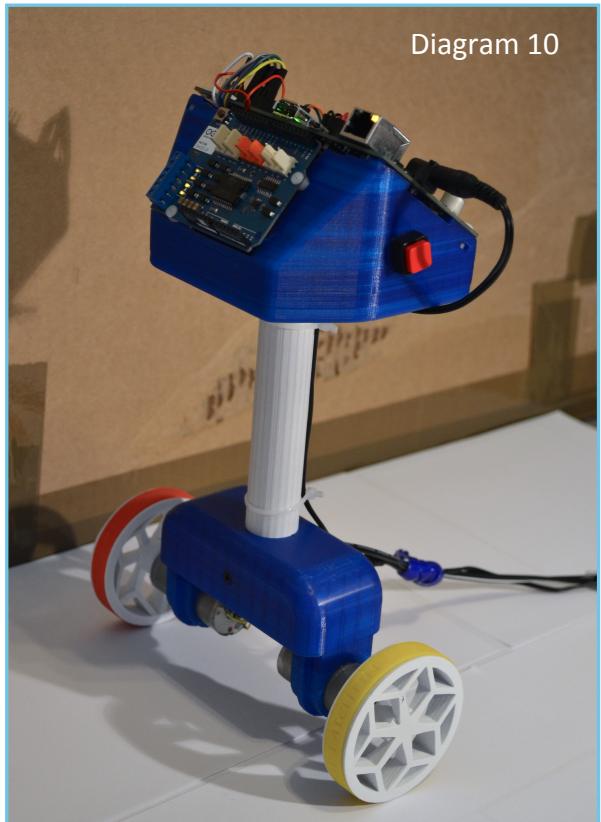


Diagram 10



PROGRAMMING INSTRUCTIONS

4.1 Computer Software

The Galileo can be programmed either through the customized Arduino IDE, or directly using the Linux interface. For this demonstration, we will be using the Arduino IDE. The specific software can be downloaded from **Intel ARK Download Centre**:

http://downloadcenter.intel.com/Detail_Desc.aspx?DwnldID=24000

Please follow the installation instructions described in great detail here, on the **Getting Started Guide**: <http://communities.intel.com/docs/DOC-22796>

If this is your first project with the Galileo Board, here is a quick breakdown of the steps that need to be done (otherwise skip to the next section):

- Download the software
- Install the relevant drivers
- Connect the Galileo via the ‘Client’ USB port
- Select the relevant board and port in the IDE
- Update the board firmware

4.2 Installing Libraries

Before we proceed to compiling and uploading the self-balancing sketch, we need to import the proper libraries. Otherwise the sketch will not compile. Provided with this document will be a zip-file called “MPU6050”, and one called “I2Cdev”.

- To import this library, open the Galileo Arduino IDE, and click: **Sketch > Import Library... > Add Library...**
- You then have to find the zip-file “MPU6050” in the finder window, and click **Choose**.
- The library should now be imported and ready to go!
- Repeat the above steps for the zip-file called “I2Cdev”.
- Please note that if you have multiple copies of the Arduino IDE (One for Galileo and one for normal Arduino use), the libraries will still import into the same destination folder (Documents > Arduino > Libraries).

4.3 Uploading the Balancing Program

- Open up the provided Self-Balancing Robot sketch in the Arduino IDE.
- Make sure that you have connected the board as described above. Then compile and upload the code.
- The sketch should start running straight away, but it takes around 20 seconds for the accelerometer-gyroscope module to auto-calibrate. Only then can the stabilisation procedure start!

NOTE: The program needs to be uploaded again each time that the Galileo is turned off and on again! *I am currently working on a newer version which works through the Linux Shell, where I hope that this problem can be bypassed...*

4.4 Optimizing the Performance

- The program has a built-in debugging system. To use this debugging system, open the serial monitor of the Arduino IDE after uploading the program.
- The program uses a PID controller in order to balance. Depending on a variety of factors specific to the balancing robot (weight, motor speed, size, etc.), the PID values may need to be changed slightly to increase the stability of the robot.
- If you built the robot to the exact specifications provided in this document, the provided PID values should work quite well. To supply your own values, either change them in the actual Program Sketch, or supply them through the serial monitor. For example, you can type the following into the serial monitor:

```
P4.9, I0.59, D3.25
```

The Galileo should then say `Constant set: 4.9` to confirm that it received the new values.

- Fine-tuning the PID controller is tricky, and requires a great deal of trial and error. The aim is to set the **P** and **I constants** quite high, to give the robot a faster response rate. The **D constant** is then increased slowly to dampen the robot and reduce any overshoot.

DEMONSTRATION AND USAGE

5.1 Pre-demonstration Checklist

Once the self-balancing robot has been completely assembled, programmed and optimised as described in the previous section, you are ready to demonstrate and play with the balancing robot. Here is a short list of steps which need to be taken before showing the robot in action:

- Plug in the power supplies.
- Ensure that the toggle switch for the motors is turned off.
- Upload the sketch using a USB cable and the Arduino IDE.
- Leave the robot lying on its side and open the serial monitor.
- Once the monitor says: “Ready to Balance”, pick up the robot and hold it in a balanced position. Press the button on the side of the robot to set the robot’s centre of gravity.
- Put the robot down again and disconnect the USB cable.
- Now flick the toggle switch to turn on the motors.
- The robot is now ready for demonstration!

5.2 Demonstration Procedure

- Pick up the robot and slowly move it into a vertical position, until the motors start to work. Let go of the robot and allow it to stabilise.
- Once the robot remains in a relatively stable position, lightly push the top of the robot and demonstrate its ability to deal with the disturbance and return to its original position
- Once the demonstration is complete, turn off the motors using the toggle switch. Place the robot onto its side and unplug the power cables.
- If the robot overturns during the demonstration, don’t worry! It turns off automatically after overturning, and will begin again once you place it back into a relatively vertical position.

5.3 Post-demonstration Checklist

- Once the demonstration is complete, unplug all of the cables from the robot.
- Wrap the robot in its protective packaging and return it into its box. (If you don’t have a protective box and packaging, then get some!)
- Bash all of the cables together and include them in the box for future use. Now you are ready to depart!

TROUBLESHOOTING

6.1 Common Problems

All of these solutions assume that the USB communication with the computer is properly connected and configured, and that the Galileo is plugged in! ☺

1. The sketch will not upload and the Galileo seems to be unresponsive

This issue appears after using the Galileo for longer periods of time, or after uploading multiple sketches through the Arduino IDE. To solve the issue, disconnect the USB cable and then restart the Galileo Board. I am not sure what the exact cause of this issue is, but I believe it only appears when using the Arduino IDE, and not when communicating directly with the Linux shell of the Galileo. (I am looking into this problem at the moment!)

2. The sketch will not compile / There are compiling errors

Make sure that you have the newest version of the Galileo-Arduino IDE installed, and have the newest firmware installed on the Galileo Board. The Self-balancing Robot requires the following **Arduino Libraries** to be installed in order to compile properly:

- I2Cdev Library (Manually Installed)
- MPU6050_6Axis_MotionApps20 Library (Manually Installed)
- Wire Library (Comes with Arduino IDE)

If all of the above libraries are not installed, then please do so now, as described in the [4.2 Installing Libraries](#) section of this document.

3. The Sketch is running properly, but the self-balancing system will not start

The balancing procedure will only start once the auto-calibration of the accelerometer-gyroscope module is complete, and when the robot is held relatively still in a vertical position. (Angular Vel. ~ 0 , Current Angle ~ 0) If the stabilising system still does not start, then try resetting the centre of gravity by pressing the button on the side of the robot while holding the robot in a vertical and balanced position. If all of the solutions above do not solve the problem, then make sure that your accelerometer-gyroscope module is working correctly.

4. I am using the provided Sketch, but the robot doesn't stabilise properly

In order to stabilise, the sketch uses a PID controller to send the required commands to the motors. The PID constants required in order to stabilise the robot depend on a variety of factors such as weight, size and motor speed. If the provided PID values do not give optimal stabilisation, try altering them slightly to see if you can get a better response. PID controllers are quite advanced engineering stuff, so I am afraid the only method for amateurs to get the correct PID values is through trial and error! ☹ (Please let me know if you have a faster way of getting good PID constant values!)

This list of common problems will get longer as I come across them myself, or if people send me some feedback!

6.2 Other Issues

If you are having issues with getting this self-balancing robot to work properly, or if you have any improvements or comments you want to send me, then please contact me and I'll respond as quickly as I can! You can contact me in the following ways:

- Leave a post on my thread of the Intel Makers Forum:
<https://communities.intel.com/community/makers>
- Write a comment anywhere on my technology blog:
<http://wired.chillibasket.com>

If you are contacting me about a problem, please include as much information about your setup as possible. Pictures are also a great help!

CONCLUSION

Now that you have completed the document and (hopefully) built a working demonstration of the Self-balancing Robot, please don't just stop there! There are many steps you can take to improve the robot further. How about adding a wireless/Bluetooth module to the robot, and build a remote control to drive the robot around?

I personally am going to continue working on this project to improve it in a number of regards. I hope to make it battery powered and wirelessly controlled, so that I have my own R.C. Self-balancing Robot!
