|  |  |  |
| --- | --- | --- |
| PRODUCT SPECIFICATION DOCUMENT - PRT | | |
|  |  |  |
|  | Fon52500 |  |

Table of Contents

[1.1 Introduction 3](#_Toc440990523)

[1.2 Technical overview 3](#_Toc440990524)

[1.3 File overview 4](#_Toc440990525)

[1.4 Future steps: 4](#_Toc440990526)

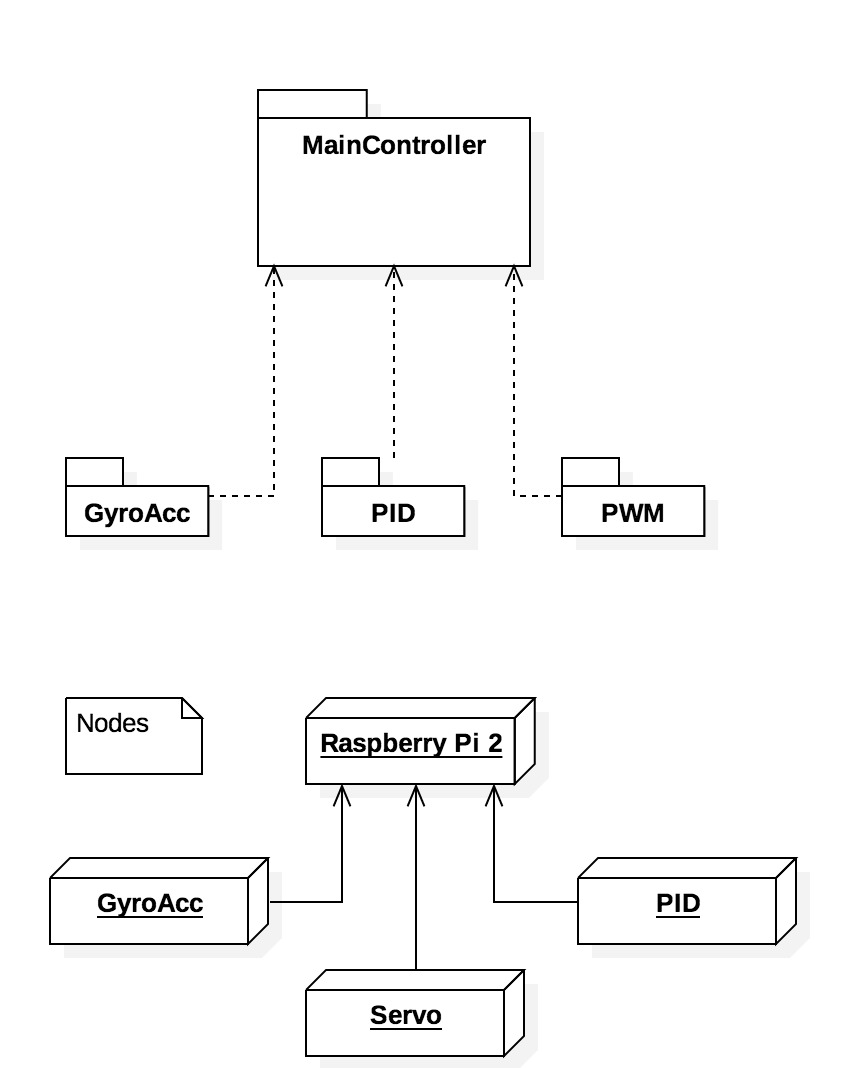
## Introduction

For the PRT project we’ve started with a project that was based on a controlling system that could balance and drive based on a linux powered device. In this project we’ve chosen to do this with a Raspberry Pi 2 with an own buildroot environment. In this document we want to show how far we’ve gone and which part are still TODO. By following this document people can find out where they can start in fulfilling the project.

The different aspects that are still todo, can be described as a wide range of competences. Some of them are easy to solve, while others will aren’t easy to solve. Also the subject of the different todo’s covers all project perspectives. From the documentation till the module layer in the kernel space of the Linux buildroot, improvements could be made. Because the project isn’t nearly finished and a lot of testing, programming and discussing could still take place we want to be sure that people can easily start with this project where we have left. For this purpose this document is written.

## Technical overview

Because we’ve worked in a group a lot tasks and programs have been build in parts. Those modules/packages can be found in the figure below:



All those parts are written in a module to work in the kernel environment of the RPI 2. Because this is one of the learning purposes of PRT this can be further integrated by creating a controller that can interface with all those standalone modules. To accomplish those steps we first have to set the correct building environment. To create those environments please follow the next steps:

**Building steps RPI 2 buildroot (Thanks to Bart Janisse)**

* Creating buildroot environment for RPI 2
  + <https://bitbucket.org/bartjanisse/raspberry-pi-2-buildroot/src>
* Make cross-compile environment from linux to the RPI 2
  + <https://github.com/bartjanisse/BeerTender/blob/master/Cross%20compileren%20van%20modules%7E>

When those steps are try one of the helloworld modules to load into the RPI 2 and see if the module is loaded into the sytem (for example: /proc/modules/)

For some reason there are some errors with the Ethernet connection to the RPI 2. Those connectivity problems seems to be only exists on the Lubuntu version distributed by Fontys. Unfortunately we have no time testing with other versions of Linux but with small adjustments this could be possible. (ping and ssh from Windows and OSX seems to be far less vulnerable to errors)

Before starting with those problems make yourself comfortable with commands like SSH and netstat/ifconfig and top. With those commands you can easily detect why the RPI is reacting this way.

(Hint: Attach a screen to the RPI 2, this works for the buildroot build and makes debugging a lot easier.)

**If connectivity with by is not consistent try to plug and unplug the ethernet cable when Linux is searching for a connection. Also manual changement of the host (Linux) could be working.**

**When raspberry is very slow over SSH please put comment characters (#) in front of the two lines in etc/resolv.conf**

## File overview

All files we’ve used and created for our product could be found on our github repo: <https://github.com/bartjanisse/BeerTender>

These files are divided over different persons. The overview below is given where most (working) parts of competences could be found:

**Jeroen: PID controller**

**Dennis: Gyro/Acc**

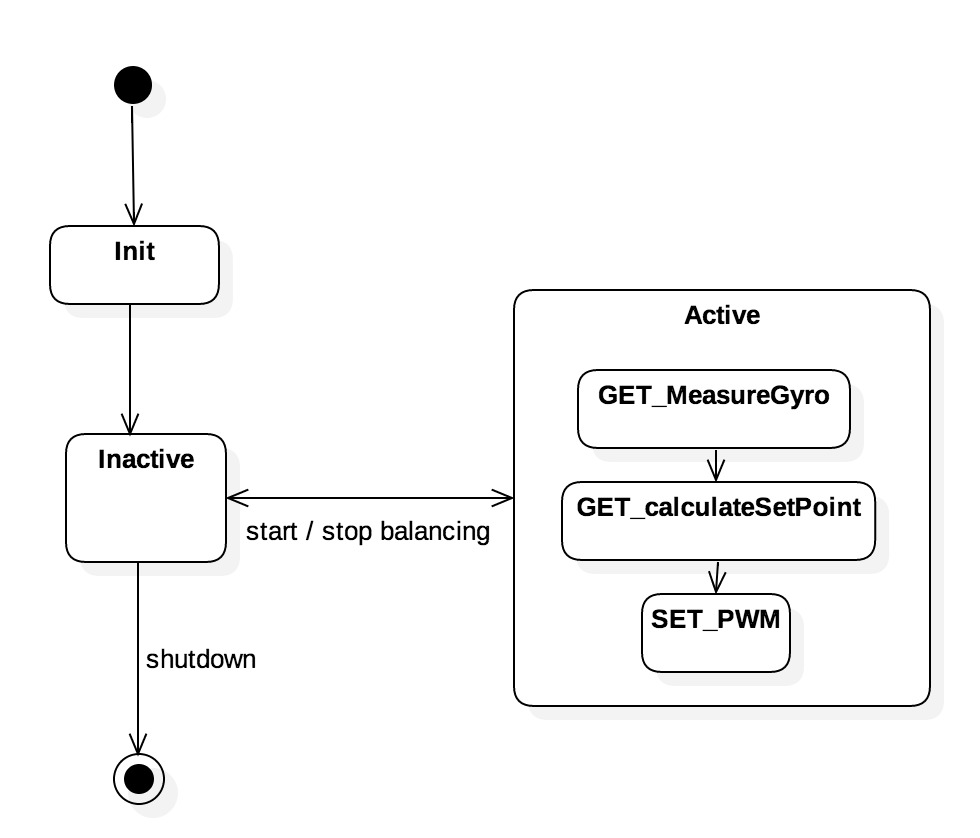
**Bart: PWM**

**Robbert Digital IN**

In those different folders information could be found on the modules as well as on normal C programs. A lot of our development has started out of C programs before we turned them into kernel modules.

## Future steps:

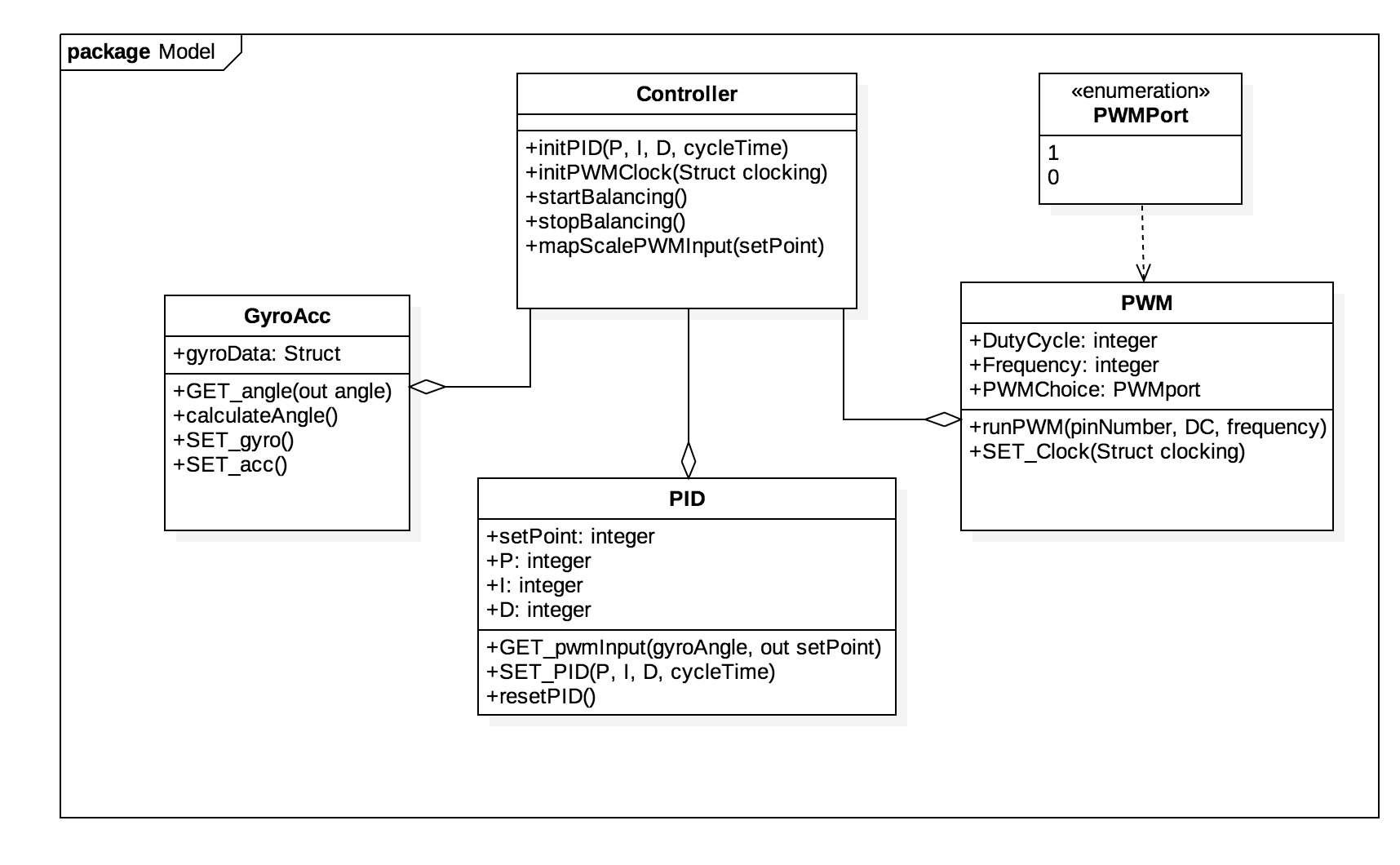
As described earlier most (kernel) parts that can be found on git are working examples. Fort those examples the controller above it is still missing. Our vision regarding the main controller can be found in the next figures:



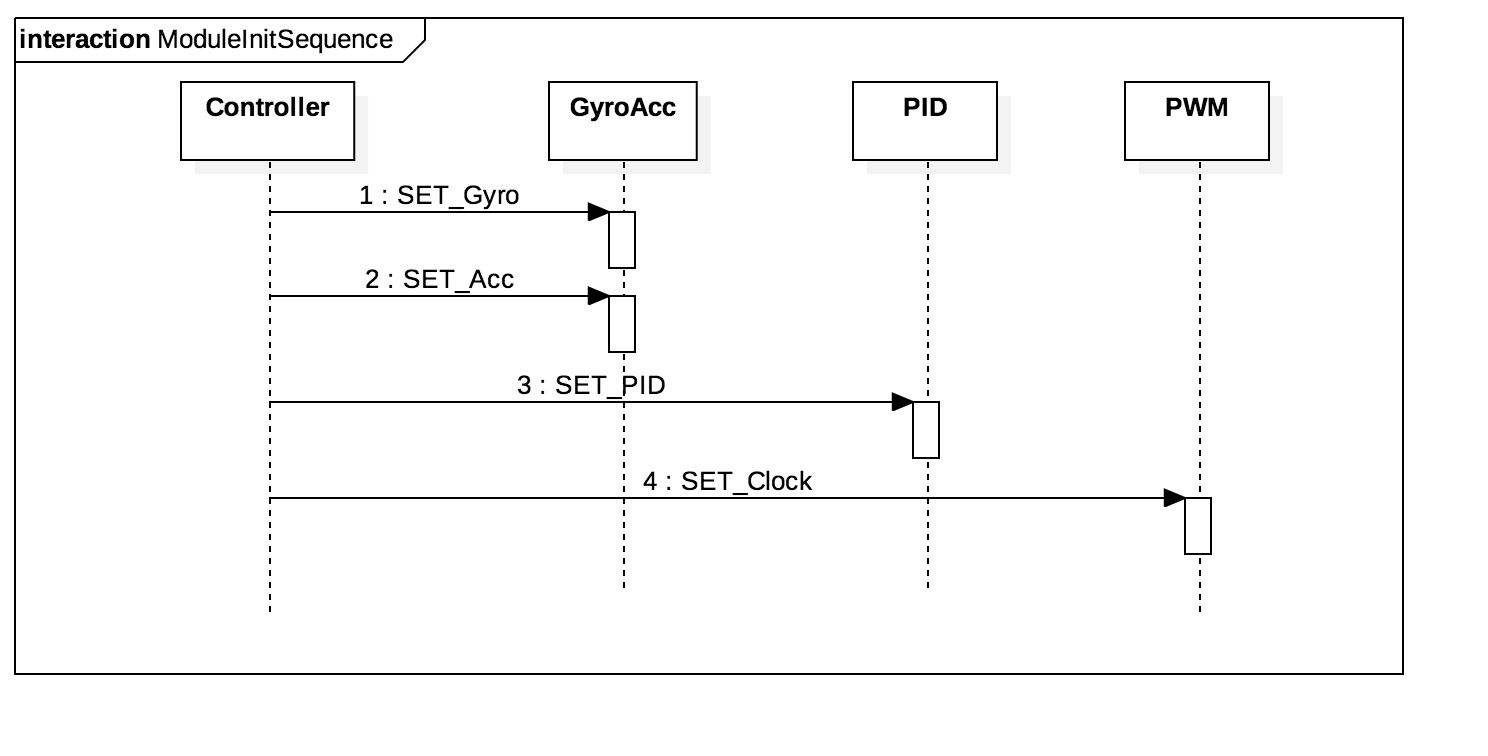
In this statediagram you can see clearly that hardware has first to be initalized before it can balance. This initializing must be done on the PWM, PID and Gyro/acc. By this initalization clock frequencies, PID parameters and Gyro/acc modes must be set. When all of those items are initalized the next step of inactivity must be triggered. This is the state where the robot is still not moving but is ready to balance. When an start trigger is given the main controller must read and process the values of the different modules to balance the robot.

Eventually when the stop trigger is given the robot must return to an inactive state. This inactive state must keep the initialization steps until it’s completly shutoff. In the future the definition fort he different states could be enhanced by testing a automatic initialization stage so the robot could chose to initialize itself and prevent the robot from drifting in by the modules. Some kind of log regarding self learning of the robot could be implemented. All of those extra possibilities will be available for implementation when the robot’s firts initializing is done. From the basic steps of the robot we can build on to create a self learning algorith that can optimize the way of balancing by analyzing data. This future perspective if far away from the current scope.

For show some other detailed information about our vision we’ve created sort of an class diagram to show how elements could interact with each other. Most of those items are already been described in the previous parts. This is the overview of the upper level class diagram. In the image below you can find an overview of this class diagram:



As discusses earlier this implementation is not integrated on GIT on this moment. This is just a sketch about how we think the program should work. If it exactly will work like this isn’t confirmed so only testing and combining the modules can tell.

Our last important vision on the robot is the sequence diagram. In this diagram we want to show something about how the robot should act on the different commands. Those commands could be divided into two different parts. First the commands that will be given when the robot needs to be initialized. This initializing state is focused on setting the different parameters of the modules. When all those functions are called the different modules will be implemented and loaded into the kernel ready to start. An overview of these initialization commands can be found in the sequence diagram below:

Besides the initial commands the robot needs to ask the modules for different kinds of information to stay in balance or to move in a given direction. This movement and balancing is out of this scope for this stage of the project. What the commands do show is the flow of information that is needed for input in the next module and what the output of the previous model will me. As can be seen in in the sequence diagram the information is flowing from the starting point (Controller) to the Gyro the PID and at last the PWM. This information flow must be secured in the main of the controller. From our perspective, writing the main of the controller and let it interface with the different modules will be the first set point in continuing this project. The sequence diagram as described is figured below.

