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

Table of Contents

1.1 Introduction 3

1.2 Voor wie 3

1.3 Projectdefinitie 3

1.4 Uitvoerders 3

1.5 Succesfactoren 3

1.6 Projectuitvoering 3

1.7 Taken en rollen 4

1.8 Verantwoordelijkheden 4

1.9 Contact 4

1.10 Overige definities 4

1.11 GIT 4

1.12 Focus 4

1.13 Sprint rapportage 4

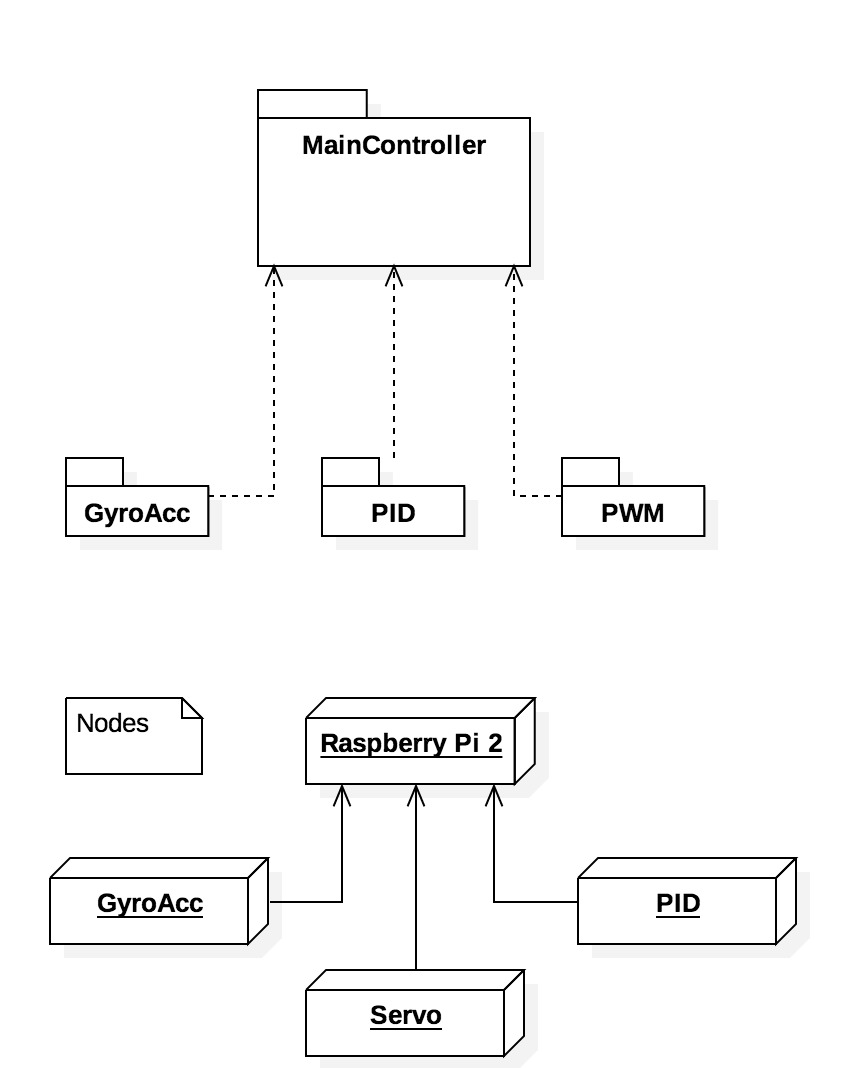
## Introduction

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.)

**BART: SOLUTION Connectivity? (ik heb die van mij altijd gewoon laten zitten ☺)**

## 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**

## 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:

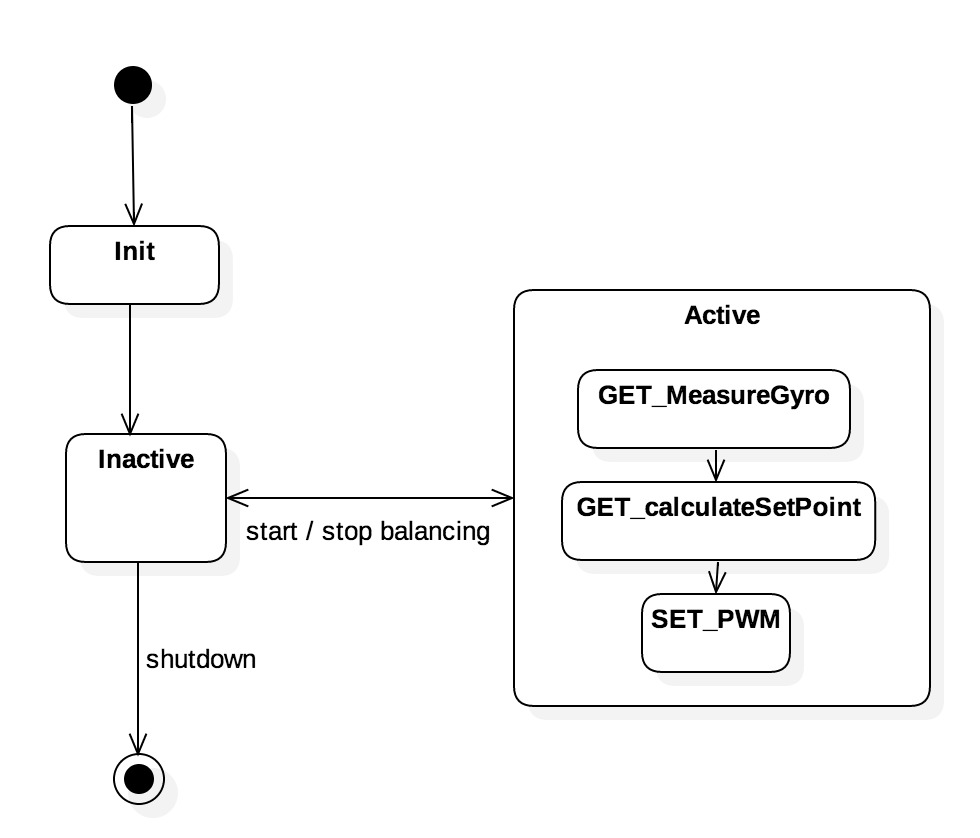


Figure : StateDiagram

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 active. This is the state where the robot is still held 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.

