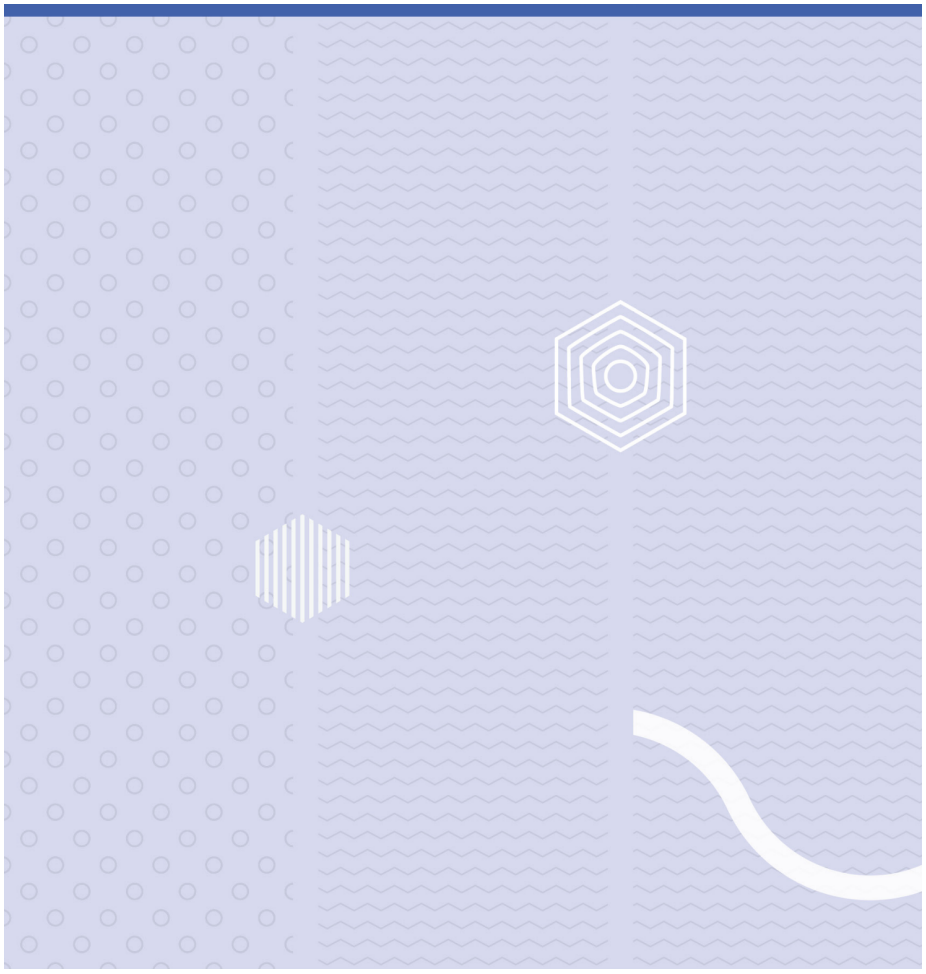
test

Hello



Title

Kent Odde, Stian Onarheim, Tarald Vestbøstad

October 5, 2020

1 Maybe useful text

Thursday 20. August 2020, the group gathered and had a meeting with Steven Bos. We discussed our project idea and the potential use of the Microsoft HoloLens 2. It was in the group's best interest to put our resources into the microcontroller rather than the goggles, dismissing any development with the HoloLens.

On September 10. the group met to plan the project. After deciding on the initial design, we drew a sketch and wrote a short description. This was sent to our professor, along with a list of required parts as well as the associated budget. The initial design can be seen in section XX.

On the 16th of September, we ordered three Arduino Nano 33 BLE Sense w/headers from Arduino.cc. This came to 1080NOK. In addition to this the professor provided us with a car for the project. It is called Turnigy Trooper, and was originally a radio controlled car. However everything but the battery and servo motors for thrust and steering, have been stripped away.

The servo motors are controlled by a pulse-width modulated signal. This means in essence that we send discrete signals, where the signal will rise at a fixed frequency. The length of time the signal is high before going low will determine the behaviour of the servo motors. Luckily for us there is a standard regarding the behaviour generated by a specific pulse width.

We will need to write a library which will abstract this away in the code. Where the interface for steering will take an angle, and the motor interface will hopefully be able to take a velocity in m/s, where negative numbers will mean going backwards.

After Steven showed us a proof of concept that it will be quite doable to base our library for the Arduino Nano on the existing toolchain for NRF52840, we started developing on the 17th of September. However there quickly emerged a problem, as Gnat arm-elf initally only supports ZFP-runtime libraries for this card. This meant that it would only support barebone functionality, and we would never be able to make use of the Ada realtime library, or even

multithreading the software. Luckily, there exists support for full ravenscar run-time libraries for the card. We followed the official guide [kilde]`https://github.com/AdaCore/bb-runtimes` to generate and build them ourselves.

On the 18th of September we improved our library build, by letting the project_wizard script in Ada drivers library generate the gpr-files for our board automatically. All we had to do was to provide some basic information about the microcontroller, and the script did the rest. We also started adapting the pin mappings from the nrf52832 to the nrf52840.

Provided that our efforts lead to a useful library, we will create a pull request into the Ada drivers library, so that others may benefit from our work.

October 5.

We received the Segger J-Link debugger from our professor. In order to get this to work, we installed the firmware from segger.com. However we also had to install a pack for pyocd in order to find the debugger:

pyocd pack –install stm32l476VG

As all development within the group is done on Linux, we also had to add udev rules, so that we were able to access the usb ports without being root. These rule-sets were found on pyocds git repository.

We also had to give the respective user access to the serial port by running the command: usermod -a -G dialout MY_USER_NAME

Legge til utilization test og grafikk.

2 Ravenscar

The full ravenscar run-time library offers what is called the Ravenscar profile. This is a subset of the Ada language, where the features and available libraries are limited. The reason for this is to ensure robust software on realtime and safety critical systems.

The specifications of the ravenscar profile are the following: https://www.ada-switzerland.ch/rm/RM-D-13.html

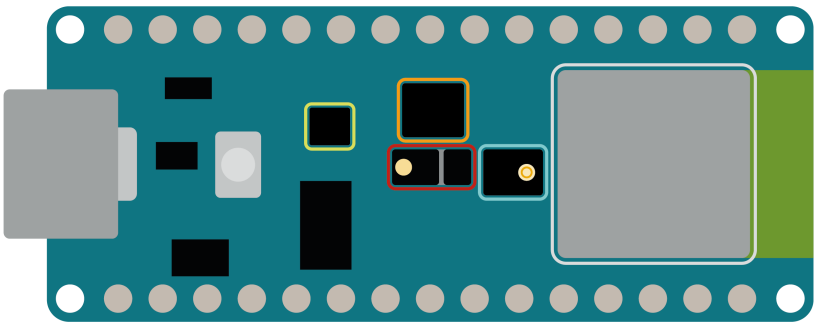
There are a couple of limitations here that will affect the way we initially planned our software.

The maximum number of entries in tasks are set to 0. This means that we won't be able do have dynamic tasking, and all tasks will rather have to be defined at compile-time.

Another thing we planned was to specify which tasks would run on the respective cores of the CPU. This however, is not supported within the ravenscar profile, and we will need to do all the scheduling as if we only had a single-core processor.

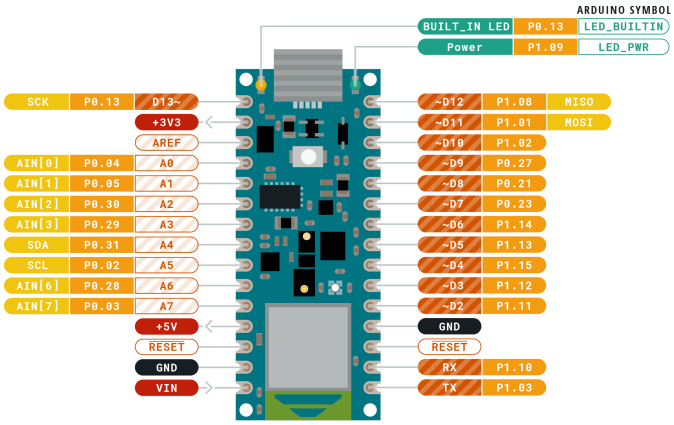
3 Maybe useful pictures

NANO 33 BLE SENSE



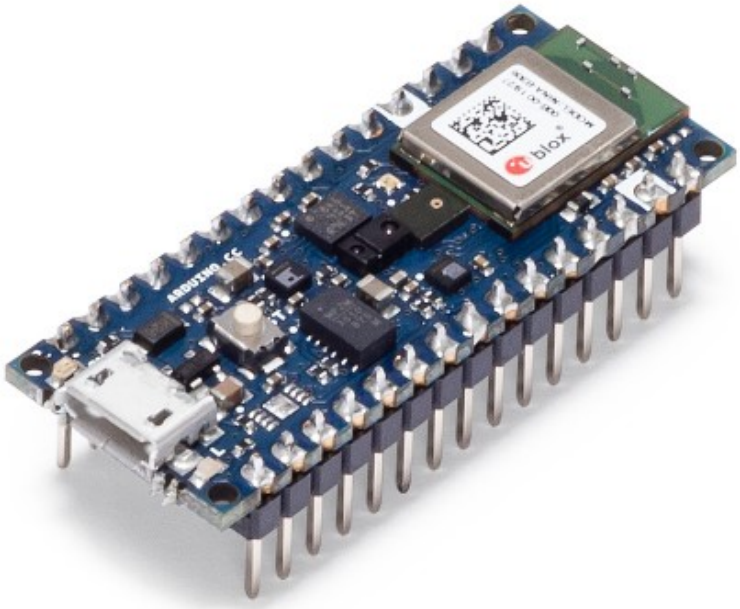
- Color, brightness, proximity and gesture sensor
- Digital microphone
- Motion, vibration and orientation sensor
- Temperature, humidity and pressure sensor
- Arm Cortex-M4 microcontroller and BLE module

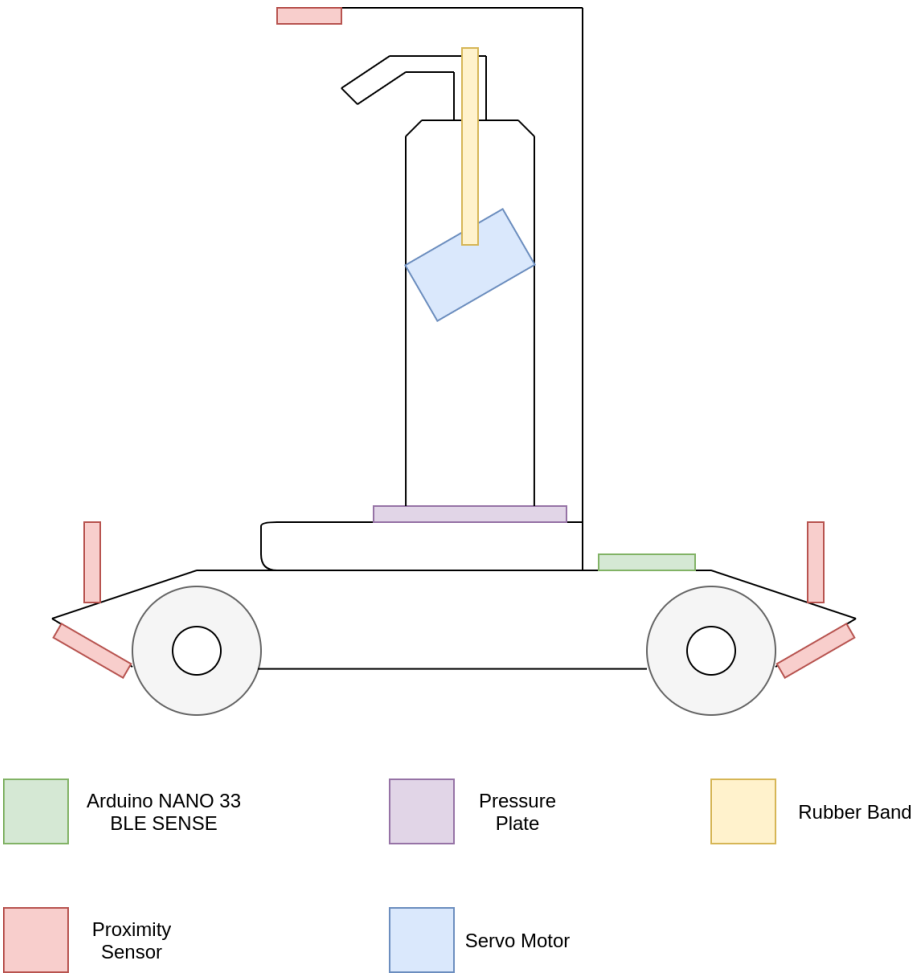
ARDUINO
NANO 33 BLE SENSE



Ground
Power
LED
Internal Pin
SWD Pin
Other Pin
Digital Pin
Analog Pin
Default
Microcontroller's Port

ARDUINO.CC


ARDUINO.CC
blox



4 Maybe sources

https://store.arduino.cc/arduino-nano-33-ble-sense-with-headers
https://www.microsoft.com/en-us/hololens/buy
https://github.com/AdaCore/bb-runtimes