Integration of a new camera

An insight into the integration process of a new camera, how, when; and most importantly: why.

Thomas Schenk

Contents

[Why a new camera? 3](#_Toc131107677)

[Installation of standard packages 3](#_Toc131107678)

[Integration on the JACKAL 4](#_Toc131107679)

[Comparing feed outputs 5](#_Toc131107680)

[Broken camera 5](#_Toc131107681)

[Conclusion 6](#_Toc131107682)

## Why a new camera?

The current problem is that the camera feed generated by the camera is very slow/choppy. This feed is not reliable at all for first person driving or accurate mapping. Let alone being able to monitor or recognize anything going on when far away from the robot. Furthermore the PO gave us a requirement which requires the robot to be able to see in the dark, the current camera did not support IR camera modes and therefore does not honour the requirements.

We hope to improve the above mentioned problems by implementing a new camera. A faster more stable speed allows us to drive the robot much more efficiently when driving first person. And also, the IR functionality like mentioned previously is a reason why this camera is substituted.  
This will serve the operators of the robot very well.

## Installation of standard packages

The installation of the camera went near flawless, Intel provides really good documentation on how to install their products.

I first installed their proprietary software to simply check if all the outputs were correct and worked like they should.

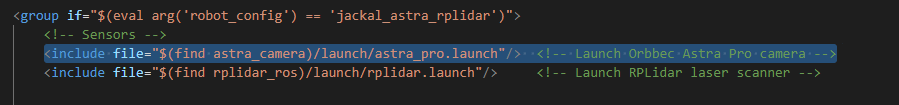
After this I downloaded the ROS wrapper and tested it with RVIZ, this proved to work too. The next step was integration on the JACKAL.

## Integration on the JACKAL

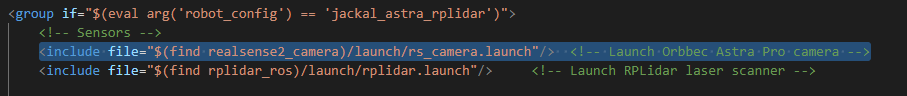
ROS works off launch files, to launch something; execute a certain launch file. When previously testing, this wasn’t relevant. We needed one ROS launch file for the camera and that’s it. But on the JACKAL, we have to take into careful consideration that there are about 10 other launch files, all which need to respect each other. In order to fix that, we need to slip the launch file of the new camera on the exact place where the old one was located. We do this as follows.

At location: AEB\_Repo/src/autonomous\_exploration/launch/exploration.launch

This is the location of the “exploration.launch” file, the main file which groups all launch files together to initiate the exploration task of the robot. In the following piece of code, the camera directory is set.



We need to change this line in order for the software to find the camera launch file. The nice thing about ROS integrated CPP code is the fact that we can use the find command to specify ROS wrapper directories. After changing, the exploration.launch file looks like this:



Rs\_camera.launch is the most basic of launch files for the intel realsense camera series, if further down the project a tweak is needed; this tweak will be linked to exactly this file again.

For the sake of readability and the fact that this is a very clear action, I will not indulge in the explanation of every single tweak there might be in the future.

## Comparing feed outputs

To validate this outcome, we can simply view the infrared camera stream and see if it outputs something which is not a black screen. This happened to be the case. The requirement was fulfilled.

We also wanted to fix the latency issues. To test the speed correctly, I needed to recreate the scenario like it will be when the end users use this product. Which is remotely via some sort of data tunnel. In my case, SSH is the option I went with. I tested the speed of the feed by having colleagues walk and perform quick jerky motions in front of the robot. If I could distinguish in a reasonable reaction time if there was a person or what they were doing; the speed would be adequate. This happened to NOT be the case. Sadly, the choppy feed was not due to the internal speed of the camera.

When testing the robot for its primary application, driving from first person view. The feed sadly wasn’t any better. Though it was actually doable to drive it, with little pauses in between it was no problem to navigate through the building.

*I would have liked to put pictures here of how the streams looked, however due to the camera breaking (see next chapter) I did not have the time to do this. I have showed all the teacher it works, yet currently it is logistically not able to put pictures of proof in this document without “lying” and taking one from the internet. This will be resolved in the next sprint.*

## Broken camera

While all above is true, I cannot validate it anymore on the system of the robot. This is because we found out our camera is broken. In rare instances, these intel cameras break. The symptoms are that the software detects the camera as USB 2.1. Which limits data transfer speeds and thus functionality.

I checked every possibility in the settings of the JACKAL, some programs recognized the camera as 3.0 properly, while others only recognized it as 2.1. It seemed to randomly decide whether it was a 2.1 or 3.0 connection.

Online this problem seems to be very frequent, with the solution being simply to return the camera to Intel. We fabricated our own “solution”, which was to get another intel camera and check if it worked. Like magic, this camera WAS detected as a 3.0 device; completely confirming our hypothesis that the current camera was broken.

## Conclusion

Concluding we can state that this integration of the camera was, on a technical level, a success. It can not only be tested that everything works; but also proves very insightful towards possible future integrations.

Yet when conforming to the initial problem this upgrade was trying to solve, it only did 50% of that. We integrated a fully working IR stream into our product, which was a customer demand; very nice! However, we did not resolve the choppy/laggy stream. I did some research off the charts about this, and this behaviour is often linked to a CPU overload. Which can spark another research into this problem.