First Person Driving

Contents

Introduction

The controls of robots have been an ever-evolving phenomenon over the years. While at first, we were tethering them to a bunch of wires, with only a joystick to control; we now can let them do the most advanced tasks on their own.

Vision still is one of the hardest hurdles to overcome when it comes to robotics. Many robots are past the point of the need for human eyes. Though a niche group of robots still rely heavily on them. This group can be described best as a group of “competitive” robots. Like sports robots, or racing robots.

Our sports robot still has a need for human eyes. Commonly, these eyes are from the side of the field. However, we would like to upgrade our eyes, to a first-person perspective. Essentially being a football player on the field, through the eyes of the robot.

Problem

Currently there are **no** visionary sensors on the robot, meaning that the robot cannot process any imagery. First person driving is therefore rather impossible.

Solution

Find a near instantaneous method to display a first person perspective from the robot to the user. This can be in the most simple way possible, while still adhering to the requirements.

Requirements

There are a few requirements which the solution to this problem must adhere to. Below they are listed.

- Provided hardware

The PO provided a stereo camera which should be used in the solution of this project. It isn’t a requirement to specifically use the model camera given; though the use of a stereo camera is.

A stereo camera is a “dual” camera, which provides a stream of both the camera feeds combined. This way, it can be interpreted as two eyes.

- End-point of the solution

Naturally, the camera stream points to an endpoint. The PO told us this endpoint can be as simple as possible, even preferring it to be just a simple web browser.

- Usability of the solution

First person driving is an amazing feature, though comes with a major concern; speed. The speed of the camera feed can be detrimental to its usability.

The PO requires us to provide a solution which is near instantaneous; due to lack of research and knowledge there are no set boundaries or metrics to adhere the final product to. Though we were allowed to judge whether we think it was sufficient or not. Naturally, this is to be validated later.

- Linux compatibility

The robot is ever evolving, especially in this current state. During this improvement, there has been lots of talk about adding a Linux machine to it for various tasks. Since camera processing is a heavy load, the PO told us that it would be beneficial to put this load on the Linux machine.

Proposed solution

Considering all the requirements:
Find a way to connect and extract imagery from the stereo camera on a Linux distribution. After this, host the stream to a local network. Not only is a local network the most like the real life scenario, it would also ensure the fastest transfer speeds. The stream can then be visualized over a http connection.

Progress

Below described is the process to reach a solution to this problem.

Camera information

As mentioned in the requirements, a stereo camera was already given to us by our PO. Alongside the information that this was an “AliExpress” camera. Initially this complicated research for datasheets and information, though quickly we could find the camera and its information online.

If we take a close look at the circuit board of the camera, we can read its serial numer/name:
“*GXIVISION-LSM36156*”

This brings us to the store page:
https://www.ebay.com/itm/355093813100?_trkparms=amclksrc%3DITM%26aid%3D1110006%26algo%3DHOMESPLICE.SIM%26ao%3D1%26asc%3D257779%26meid%3Ddde20a05ff844c639f7717aa097b8c51%26pid%3D101875%26rk%3D1%26rkt%3D4%26sd%3D355026052546%26itm%3D355093813100%26pmt%3D1%26noa%3D0%26pg%3D3458402%26algv%3DSimplAMLv11WebTrimmedV3MskuWithLambda85KnnRecallV1V2V4ItemNrtInQueryAndCassiniVisualRankerAndBertRecallWithVMEV3CPCEmbeddingSearchQAuto&_trksid=p3458402.c101875.m1851&amdata=cksum%3A355093813100dde20a05ff844c639f7717aa097b8c51%7Cenc%3AAQAIAAABYMF6rapl24zPGkjYHfiSlLTuBicNRv1Nr6Iohq4bMvd7DlxPac1R4%252FnhV6ZdsOB3e44VPDg3LMjzKh6h9F%252BnyuldWK33VEcBpK6UyRXA9oDADzwEJIB%252BjrUQktYR0EluQdk0jMCISI62ak4peNZttnRbSFGqwdC47qFkTZC5xxTjQzIEVh6Mc7jrpT1UmqndwkhWzjNTJlpYTN5OJEPMzPe2Jnwp2J%252F6HT3CDjjQGZwQVySnbOimidn4y822fst%252BO0Q6KsZSuvtFYDMAFosrIW5GbfoRil3FmWl3GNJc9p3RXfKAIQnE9SFFyt1KSocraOpjOSExPr63vjszt31jsyY844Fs82Xu4tFGFocHyXjYO5yyC4get4O0lcM99BDyqWfzxVKnPIzzoXy808Qh8GqKxSigVBwUUMAow5UHJkyIR%252BkFBtxZqOtNsut5B3jn05fqgDRxo8OrWFG%252BPaR1xIo%253D%7Campid%3APL_CLK%7Cclp%3A3458402

Please note that there are different lens setups, ours is the “3mm 90 degree” version.

All cameras come with detachable sides, and a mini-USB connector.

Camera interfacing

According to the seller's page, no additional packages are required to run this camera. Even more so, they claim it is a "plug and play" device.

However, to gain more insights into the specifics of the device; and prepare for future package compatibility. We installed a very common Linux package which handles camera information: Video for Linux (V4L2).

With this command we can check information about the connected camera devices, as well as their specific resolutions.

We can install V4L2 by:

Sudo apt get install v4l2-utils

We can see all camera output devices by:

V4l2-ctl –list-devices

We can check all available compression options and fps outputs for each device by:

V4l2-ctl –list-formats-ext

To quickly validate that our camera is also outputting some useful streams, we can open it with the "cheese" application; basically just a camera application for desktop use.

Streaming packages

To serve the end user with a camera stream, we need a software which takes that stream and converts it to a usable/sendable one. There are various packages out there which do this, all which have specific needs and or outcomes. We tried a few, which are listed below

Motion

Motion is an open source package which boasts a lot of impressive features regarding parameter tuning of the output stream.

Motion is great when it comes to its tunability, but this tunability makes it hard to find a good set of parameters to stream with. This made it that upon writing this document, we revisited motion to find that another set of parameters worked way better than before; essentially setting us up for failure.

Though motion is faster than we thought, it still is not built for high speed applications; and appears a bit blurry at times. It also wasn't able to run at full quality at full resolution.

Source: https://github.com/Motion-Project/motion

FFMPEG

FFmpeg is a framework created to easily capture audio and video from designated sources, like USB cameras. They boast a surprisingly simple CLI interface, which is in stark contrast to their many options.

Though FFmpeg **can** provide a compressed video stream, it does not take care of the handling of it. Something many other streaming packages do. This makes it so that FFmpeg can easily stream to programs such as VLC; but not to a bare html page. This ofcourse, since VLC handles incoming connections in such a way that the data can be parsed to a stream, html does not (natively) do this.

It is possible to utilize some javascript to parse the incoming data. While it does create a stream, it's very laggy; and does not display the stream well at all.

Source: https://ffmpeg.org/

Ustreamer

Ustreamer is a one man open source project, still regularly maintained. It takes a bit of both from both Motion and FFmpeg to create a lightweight solution which cuts the massive overhead introduced mainly by FFmpeg.

Ustreamer compares itself to mjpeg-streamer, a similar service though more focused on the longevity and storage solutions concerned with long security/CCTV streams.

Ustreamer provides an insanely fast speed, comparable to realtime. It does this by getting rid of most of the overhead messages and focussing solely on speed; rather than adding features like a timestamp.

Source: https://github.com/pikvm/ustreamer

Package choice

Due to the development of a quality monitoring system down the development line of this product, the initial choice for a package differs from the final choice. This decision was made to aid progress and scalability of this quality monitoring system.

Firstly, we opted to go with Ustreamer, instead of Motion; mainly due to the performance concerns. It has some benefits over the runner up, motion. Which mainly is its support for a hardware encoder, something motion seems to lack.

In short, Ustreamer was made to squeeze the most performance out of every compatible camera as possible. Because of this, anything else than performance was not really incorporated into the package. This made it a very ridged and unchangeable design, which would require rewriting a major part of the code to make it even a little suitable to work with.

Because of this, we switched to motion, which ultimately could provide a steady 30fps stream. This came at the cost of a little bit of quality, though not noticeable enough where it wouldn't be fit for use.

Because of this, Ustreamer clearly is superior when it comes to detail and frames per second; this is why we chose for Ustreamer.

Installation + usage

Installation is as simple as cloning the entire git repository; found here: https://github.com/pikvm/ustreamer

After successfully cloning, run the *make* command.

To quickly launch Ustreamer on defaulted settings, use the following command:

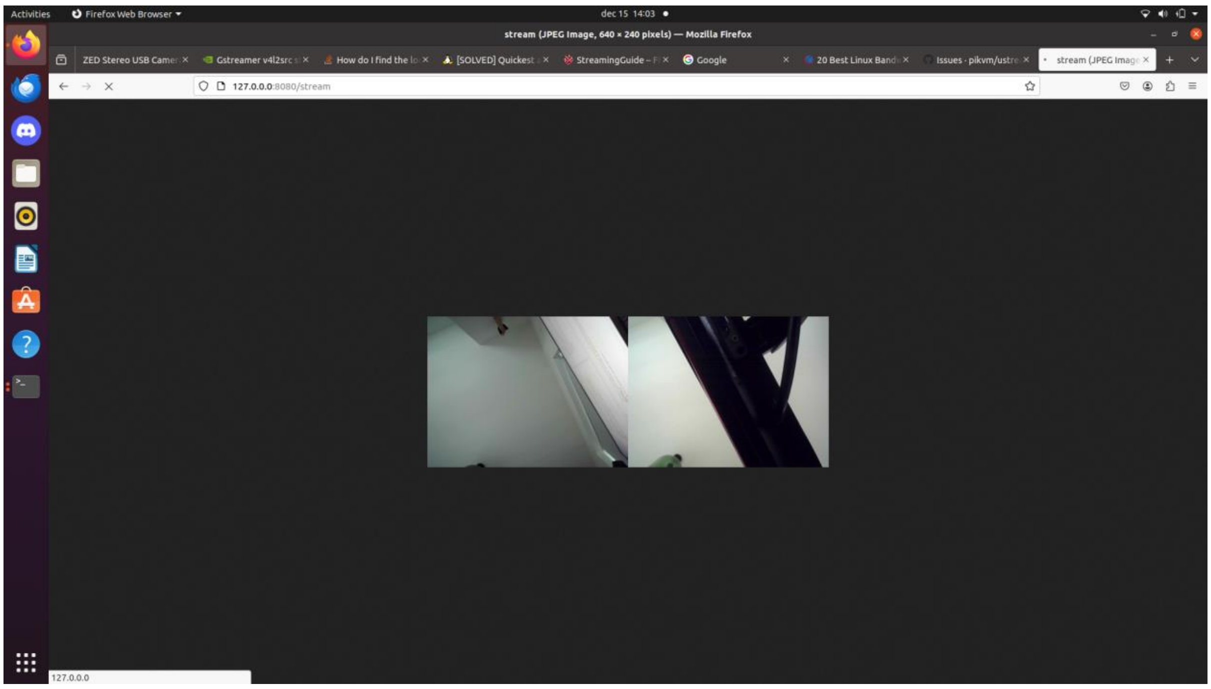
./Ustreamer

Ultimately, the setting that worked the best for us provided the following startup command:

./Ustreamer –device /dev/video2 –desired-fps=30 –device-timeout=5 –resolution=640x240 –format mjpeg –verbose

Naturally, the resolution is a variable input here.

The output picture is then shown in a webbrowser, this webpage can be edited to demand.



Future work

Continuing the work done to provide a first person driving stream, there still are a few things missing to make this a solid product.

First and foremost, the robot currently does not have a Linux machine; though an easy task, it is one to still add.

Secondly, the stream is susceptible to internet load. This way, increased network usage can slow down the stream. This needs to be addressed.