Dynamic Resolution

And its uses in First Person camera applications

Contents

[Introduction 3](#_Toc153749653)

[Problem 4](#_Toc153749654)

[Proposition to solve problem 4](#_Toc153749655)

[Process 4](#_Toc153749656)

[How to measure a slower stream/network? 4](#_Toc153749657)

[Measuring network statistics 5](#_Toc153749658)

[Latency 5](#_Toc153749659)

[Packet Loss 5](#_Toc153749660)

[Throughput 6](#_Toc153749661)

[Bandwidth 6](#_Toc153749662)

[Chosen metric 6](#_Toc153749663)

[Measuring client side FPS 6](#_Toc153749664)

[How to change the streams quality dynamically 8](#_Toc153749665)

[Relevancy of sacrificing quality for data compression 8](#_Toc153749666)

## 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

To fetch our camera livestream to utilise it for first person driving, we use the internet. Internet has many benefits, though downsides also are abundant. One of the downsides to internet is, that it can become unstable very quickly.   
This poses a risk to our camera stream, since it could likely be negatively impacted by an unstable network.

Luckily, there are systems in place to lessen the load a camera stream puts on a network; though this is not yet in place in our system.

## Proposition to solve problem

Create a system which reduces the load the camera stream puts on a network. This can be done in various ways, though the underlying outcome is that the amount of data sent over a set period needs to decrease in contrast to the amount of data sent previously.

Both the and the adjustment of the amount of data sent should be in a feedback loop. This ensures that the system is always online and can work without any manual input.

# Process

When orienting how to solve this problem, it is determined that we can best split the problem into two sub-problems.

1. How to measure a slower stream/network?
2. How to change the stream resolution dynamically?

# How to measure a slower stream/network?

The quality of the stream and its speed are roughly determined by three major factors:

1. The speed of the stream algorithm
2. The speed of the hardware
3. The speed of the internet/bandwidth capabilities

The only factor which can influence the speed of a stream unpredicted, is #3.

Measuring network statistics to determine if a stream is slowing down or speeding up is therefore a smart choice when it comes to predicting stream speed.

However, there is another way to measure the speed of a stream. FPS.

The client of a stream receives the frames of said stream in quick succession; how quick this succession is, determines the framerate of a stream.

We can measure the amount of frames received per second on a specific client, this way; we can measure exactly how fast the stream is received on the client. Especially since we know our target FPS.

To summarize, there are two main ways to predict/measure a stream’s performance:

1. Predicting how a stream would behave in the future by measuring network performance
2. Measuring the amount of frames received on the client side to determine the streams performance

We utilized both these solutions to make the final product as diverse as possible. Below is the timeline of progress on both these solutions.

## Measuring network statistics

There are various network metrics which are relevant to our stream, all of which have influence on the performance of the stream. These are listed below shortly.

### Latency

Network latency is a term used to describe the time it takes one data-packet to go to its destination, and back. The time it takes this packet is described in milliseconds.

This information is not always useful but can negatively impact communication protocols which wait for an ACK-signal to progress in their control loop.

Latency cannot be measured over the entire network. We cannot say “The networks latency is x”. This is because latency is always between two points, the sender and receiver.

We measure latency by sending a pre-determined amount of bytes to a certain destination. When we start sending the first packages, we start a timer. We stop this same timer when the receiver of the bytes lets us know they have received said bytes.

The time measured is called, latency.

### Packet Loss

Previously we already mentioned packets, in those contexts; we assume that all packets make the target destination. Though in some cases, they do not. This is called packet loss.

While previously, those packets would always more or less arrive, though just with a delay; in this case they do not at all. Which means it has more drastic effects on the application in need of that data.

To measure packet loss, we send a predetermined amount of packets (bytes) to a receiver. On the receiver, we check how many packets we received. Since we know the start amount, we also know how many we should receive.

For example, we send 100 packets to receiver X. X measures how many packets it receives, in this instance; it receives 78. This means that the packet loss is 100 – 78 = 12%.

### Throughput

Throughput refers to the amount of data flowing **through** a pipeline (both in and out!), in our case this pipeline is our network. This means you can measure throughput at many different parts of the network.

Measuring throughput is simple, but often hidden in hardware registers. It relies on counting the RX and TX bytes of a network adapter/router/network pipeline etc. Usually this is done in seconds. Observe all the traffic coming and going, and that is the throughput.

For example, a pipeline sends 16mb of data and receives 25mb of data per second. The throughput of the TX is 16mb/s, the throughput of the RX is 25mb/s.

### 

### Bandwidth

Bandwidth is a very common term. It refers to the total amount of data a network can transmit in a set amount of time. Usually, the bigger the bandwidth, the more data can be pushed through the network.

Bandwidth cannot be measured, since it is a parameter which exists outside of the control of the users. Bandwidth is “given” to a network by either its internet provider or the hardware which the network is set up with. Available bandwidth CAN be measured, though the bandwidth must be known prematurely to processing network data; which is not always the case.

### Chosen metric

All that becomes apparent is that most of the network statistics need certain data to be sent in order to recognize and calculate a significant number.

Ideally, we do not want to risk measuring a metric which might endanger our stream; by endangering our stream. That is why we want to focus our attention on the *Throughput* metric. Sadly, this does have one downside, which is that throughput can only be measured at very specific points in a network. And that its output to the first eye seems very insignificant.

To measure throughput, we have made a network measuring tool in CPP. It takes a look at the RX and TX buffers of our network adapter. After this, it measures the total amount sent; the average and the current amount sent. This can be found under the Code folder.

## Measuring client side FPS

Still work in progress, did not get around to this yet.

# How to change the streams quality dynamically

Natively changing stream resolution is not a supported feature, though nevertheless we are trying to implement it currently. There are still bugs which we are trying to phase out, though the beginning steps are certainly there.

## Relevancy of sacrificing quality for data compression

Since up until now, it was assumed that reducing the quality of a stream reduces its data output. Though this was never proven. We can use a tool called” bmon” to measure the used bandwidth on certain network adapters. Using this tool, we can prove that impeding the quality of a stream means it transmits less data.

We change the quality of the stream by downscaling the resolution.

2560x 720:

A screenshot of a computer

Description automatically generated

1280x480:

A screenshot of a computer

Description automatically generated

As we can see, even with a slight decrease in resolution; we can save 20% of bandwidth previously taken up. We could extend these measurements to include all possible resolutions with our camera, but this is not needed as of now.