TECHNISCHE UNIVERSITÄT
CHEMNITZ

Sensor data acquisition and application development with a real-time reactive data processing mesh.

Research Internship Report

Dept. of Computer Science
Chair of Computer Engineering

Ritwik Ghosh
Matriculation Nr. 484868

18th February, 2020

Contents

List of Figures

Chapter 1

Introduction

This report summarizes the tasks I performed, the things that I learned doing these tasks and the technical details of the software, hardware, technology and frameworks which I came to know, use, implement, measure and test. This chapter serves as a short overview of my activities in the research internship. The subsequent chapters will go deeper into each activity subset and explain things in detail.

One of my earliest tasks was to observe and reverse engineer the wireless UDP protocol inside a consumer sports utility camera marketed as GoPro HERO7 [1]. By design the camera would communicate with a small battery operated device called the "Smart Remote" [2] which is sold as an accessory. I used an open source tool called Aircrack-NG [3] to observe the protocol between the GoPro and Smart Remote devices. Luckily, nothing was encrypted and I was able to find the exact messages sent. Then I proceeded to implement a small library which can be used to control multiple GoPro cameras via a computer with a wireless network interface card (and with no need of any Smart Remote [2] device); essentially reverse engineering the communication protocol between the GoPro and Smart Remote devices. This library is stored in a Github repository [4].

After the above preliminary task, I was required to understand the API of Link2 [5] which is a "real-time reactive mesh for data processing" [5].

Then I developed a software interface for a GNSS [6] receiver called ProPak-V3 [7] with the Link2 [5] framework. It was important to understand the data formats developed for working with positional data and I ended up creating some data formats myself. Some offsets were seen while testing the GNSS receiver on the test vehicle and further test drives were conducted in order to understand the nature of the problem. The software developed towards completion of this task can be found in a Github repository [8].

Another task was to integrate two machine vision cameras into this Link2 [5] framework. The first was an Allied Vision Mako monochrome camera [9] and the second was a Basler Ace series color camera [10]. For both activities, it was first important to understand how the vendor API for accessing pictures from the camera works, performance issues, data rates, data formats and finally the user requirements on my side. The software developed can be found in Github repositories [11] and [12].

I was also tasked with creating applications which would detect faces and facial landmarks using dlib machine learning toolkit [13], TensorFlow [14], the real-time data mesh [5] and camera output. Camera output from the data mesh was captured

and using a machine learning model built using dlib library [15] and [16] the task of detecting faces and facial landmark features was performed. The software developed towards completion of this task can be found in Github repositories [17] and [18].

Another thing I had to learn how to do was package configuration using `conda` [19] using the GitLab CI pipeline [20]. I created a package for the Pylon SDK [21] and it can be found in [22].

Apart from software development, most of which was in C++ (and the rest was in Python), I was also responsible to make useful additions to the team user-guide [23] (you will find more information on this in the Appendix), create documentation which you will find in the respective README files in the Gitlab repositories, writing RFC's [24] for new software components.

Chapter 2

A cursory look at the reactive real-time data mesh and its developer API

"DRAIVE Link2 enables high performance distributed software systems. It is based on a decentralized and consumer driven publish-subscribe architecture" [5].

2.1 Core Components

Node In the Link2 [5] framework, a *node* is a reusable piece of software designed to solve a specific problem. A *node* serves a unique purpose or provides an independent functionality uses *pins* to communicate with other nodes on the same *Mesh*. "Nodes can be deployed independently from each other. Nodes which send messages are called publisher, nodes which receive messages are called subscriber" [25].

Pin A pin is defined as an object which acts as a conduit for messages and a link or a connection between data producing and data consuming objects (nodes, for example). It is also a point of contact which connects to other such *pins* from other nodes. There are two types of pins. *Output pins* obtain data from the node in which they are incorporated, and organize them into messages. They are also used in the sending of the data to other pins from other nodes. *Input pins* receive data objects from *Output Nodes* for processing in the node they are incorporated in. "Pins are specialists in handling asynchronous data and messages. They take care of tasks like data composition, message ordering, message filtering, quality of service and bandwidth management" [25]

Mesh A set of nodes operating together is called a *Mesh* "Nodes can join and leave the mesh at any time. Nodes are vertices and subscriptions are the edges in the mesh. A mesh resembles a decentralized application. It can run at your local computer as well as distributed on multiple hosts" [25].

Message "Communication between nodes is done via messages. Messages carry different data as payload and are being sent between the nodes using network transport (such as TCP, UDP, etc.)" [25].

2.2 Offer and Demand

This is a core concept which is needed to be understood before working with the platform. Publisher nodes provide data for further processing. Their output pins announce their available data to other nodes. The type of data which is available for subscription is called the offer. The offer is specific to an output pin. In Link2 an offer has FlatBuffers "table" format.

Subscriber need certain type of data to do the work they were designed for. This needed data is called the demand. The demand is specific to an input pin. In Link2 the demand has FlatBuffers [27] "table" format. The terms demand and data type can be used interchangeably and describe the same thing [26].

2.3 A brief explanation of the frequently used functions from the API

- Creating a mesh and discovering it on a node

```
1 DRAIVE::Link2::NodeResources
2 nodeResources { "l2spec:/link_dev/ld-node-camera-
     basler", argc, argv };
3
4 DRAIVE::Link2::NodeDiscovery nodeDiscovery {
     nodeResources };
```

  All data will be written on the mesh `link_dev` via the `push` call which will be explained soon.

- Creating a pin

```
1 DRAIVE::Link2::OutputPin outputPin{nodeDiscovery,
     nodeResources, "basler-cam-output-pin"};
```

  A output pin is created. This pin can now be used for pushing data onto the mesh. Note the involvement of nodeDiscovery and nodeResources objects in the creation of the pin.

- Pushing data onto the mesh (on line number 7)

```
1 uint8_t *pImageBuffer =
2  (uint8_t *) ptrGrabResult->GetBuffer();
3 cv::Size imageSize(frameWidth, frameHeight);
4 cv::Mat frame(imageSize, CV_8UC1, pImageBuffer);
5 link_dev::ImageT currentImage =
6 link_dev::Interfaces::ImageFromOpenCV(frame, link_dev
     ::Format_GRAY_U8);
7 outputPin.push(currentImage, "BaslerCamImage");
```

  The data `currentImage` which is of type `link_dev::ImageT` is pushed onto the mesh. Note how we also include an identifier `BaslerCamImage`.

2.4 Specification and Instance files

Link2 based components can be described and configured by two different JSON files. The first one is the specification.json file which describes properties of a node. It contains information about input and output pins, their demands and offers, data types in use and user configuration data. The specification file is being created by the node developer, specific to (a class of) nodes and the same for all instances of a node. The specification file is a contract between the node and its user [28]. The instance.json file configures the instance of a node. With instance we describe a single instance of a particular node [28]. We can use this file to define *instance specific* details which are needed for the node to run. Basically, what is otherwise known as configuration variables.

Consider a sample specification file for a *Webcam node*, essentially a node that captures data from a configured webcam and pushes all the data onto the configured *Mesh* in the below listing.

```
1  {
2    "file-type" : "link2-node-specification-1",
3    "$id": "l2spec:/link_dev/ld-node-webcam-2",
4    "pins" :
5    {
6      "webcam-output" :
7      {
8        "pin-type" : "output",
9        "offer-type" :
10       {
11         "schema-filename" : "./data/Image.bfbs",
12         "table-name" : "link_dev.Image"
13       }
14     }
15   },
16
17   "user-configuration-schema":
18   {
19     "$schema": "http://json-schema.org/draft-04/schema
         #",
20     "type": "object",
21     "properties":
22     {
23       "ImageWidth" :
24       {
25         "type": "number"
26       },
27       "ImageHeight" :
28       {
29         "type": "number"
30       },
31       "FPS" :
32       {
```

```
            "type": "number"
        },
        "Grayscale" :
        {
            "type": "boolean","default": false
        },
        "CameraID" :
        {
            "type": "number", "default": 0
        }
    },
    "required": ["ImageWidth","ImageHeight","FPS"],
    "additionalProperties" : false
  }
}
```

From the specification we can see things like the name of the *Mesh* is `link_dev`, there are no *input pins* and there is just one *output pin*. We can see that this node outputs data in the format "Image" [29] which is defined in a FlatBuffer file called `Image.fbs`. Further, we can see the configuration which is requested by this node and the ones which are mandatory for the operation of the node. We also get a lot of details about the configuration data like default values. It is also possible to enumerate range here.

In the next listing you can see a sample of an instance file for the same *Webcam Node* whose specification file we saw above.

```
{
  "file-type" : "link2-node-instance-1",
  "$id": "l2node:/my-mesh/sample-webcam-2",
  "specification": "l2spec:/link_dev/ld-node-webcam-2",
  "pins" :
  {
    "webcam-output" :
    {
      "offer-name" : "l2offer:/webcam-output"
    }
  },

  "user-configuration" :
  {
    "ImageWidth" : 640,
    "ImageHeight": 480,
    "FPS" : 30,
    "Grayscale" : false
  }
}
```

This chapter covered the basic elements of the real time data processing mesh called Link2. This was important as I will use the terms introduced and explained in this chapter like *Offer*, *Node*, *Mesh* freely henceforth.

A large portion of my work was mostly understanding sensors like a GNSS receiver, machine vision cameras, their corresponding vendor API's and figuring out how to use them. Then I had to adapt them to the Link2 framework such that they can be integrated in the overall project. For this it was important to not only understand the sensors and how they work, but also the framework for which I was writing all the software components, or rather *Nodes*.

Chapter 3

Integration of satellite navigation and inertial measurement unit

3.1 Motivation and Context

`ld-node-gnss-ins-2` [8] is a node written to retrieve positional and inertial data from NovAtel [7] OEM4 and OEMV series of devices. It is suited ideally for OEMV receivers equipped with SPAN [30] technology. The NovAtel Propak V3 [7] device communicates via a serial protocol [31]. An external library [32] is used which facilitates serial communication with the NovAtel device and provides C++ wrappers for common functions which are needed for interacting with the device.

3.2 Data profile

This node writes GNSS position estimates based on WSG84 [33] standard on the mesh. It will also provide incremental updates to acceleration and gyroscopic data. Let's take a look at the what data this node supplies. Note that the listing below follows the Flatbuffer [27] notation.

This node supplies 3 offers called GPSMeasurement, SolutionStatus, AccelerationData, GyroscopeData and they are defined as below. SolutionStatus is used to keep track of the state of the receiver. With every update of information, we also provide how much faith we have in the information via the SolutionStatus. It essentially provides the data from a structure of same name [34] inside the GNSS reciever.

```
1  //Supplies positional estimate.
2  table GPSMeasurement
3  {
4    gps_week : int32;          // GPS Week number
5    gps_millisecs : int64;     // Milliseconds into week
6    timestamp: int64;          // system timestamp
7    latitude: double;          // degrees, S<0, N>0
8    longitude: double;         // degrees, W<0, E>0
9    altitude: double;          // metres
10 }
11
```

```
12  //If the device is unable to provide a good position
13  //estimate, it is reflected in this supply.
14  table SolutionStatus
15  {
16    gps_week : int32;              // GPS Week number
17    gps_millisecs : int64;         // ms into week
18    timestamp: int64;              // system timestamp
19    CurrentSolutionStatus: int32; // Mirrors the enum
20                                   // SolutionStatus.
21  }
22
23  //Acceleration data from the device
24  table AccelerationData
25  {
26    gps_week : int32;          // GPS Week number
27    gps_millisecs : int64;     // Milliseconds into week
28    timestamp: int64;          // system timestamp
29    accownX: int64;            // Longitudinal accl.
30    accownY: int64;            // Lateral accl.
31    accownZ: int64;            // Vertical accl.
32  }
33
34  //Gyroscopic data from the device.
35  table GyroscopeData
36  {
37    gps_week : int32;          // GPS Week number
38    gps_millisecs : int64;     // Milliseconds into week
39    timestamp: int64;          // system timestamp
40    gyrownX: int64;            // RollRate
41    gyrownY: int64;            // PitchRate
42    gyrownZ: int64;            // YawRate
43  }
```

3.3 Hardware setup

The setup consists of an OEMV receiver attached to a GNSS antenna and an Inertial Navigation System(INS) using an IMU [7]. This application was tested with a setup of NovAtel Propak V3 enclosure which consists of an OEMV3 receiver, connected to an HG1700 AG58 IMU and a GPS antenna [7]. A serial to USB converter would be needed if you want to connect your laptop to the NovAtel device which is how I connected.

3.4 Development of the node

In this section I will provide brief code samples to highlight some of the important software actions inside this node.

```
novatel::Novatel novatelDevice;
```

In the above listing you can see an object of the class Novatel being created. This object is used to initialize the Novatel device, perform cleanup, hardware reset, and most importantly, the serial communication.

```
void BestPositionCallback(novatel::Position &posData,
                          double &dData);

void CorrImuCallback(novatel::CorrImu &corrIMUData,
                     double &dReadTimestamp);

void BestGPSPositionCallback(novatel::Position &pData,
                             double &dData);

void InsPositionVelocityAttitudeCallBack(
        novatel::InsPositionVelocityAttitude &posData,
        double &dReadTimestamp);
```

We create callbacks such that whenever a new segment of data is ready on the device, our implementation inside this callback function is called.

Then we *register* our callbacks like you can see below

```
novatelDevice.set_best_position_callback(
  [this](novatel::Position &pos, double &dReadTstamp)
  { BestPositionCallback(pos, dReadTimestamp); });
```

The below lines of code are essentially sending commands to control the NovAtel device [34].

```
novatelDevice.ConfigureLogs("BESTPOSB ONTIME " +
               std::to_string(1.0 / m_dataRateHz));
novatelDevice.ConfigureLogs("CORRIMUDATAB ONTIME 0.01")
```

Inside the callbacks that were explained earlier, there is code to create appropriate data packets for the mesh and write them. Note the object `GPSMeasurementT` is a Flatbuffer [27] object and push() is a function used to write real-time data on to the mesh.

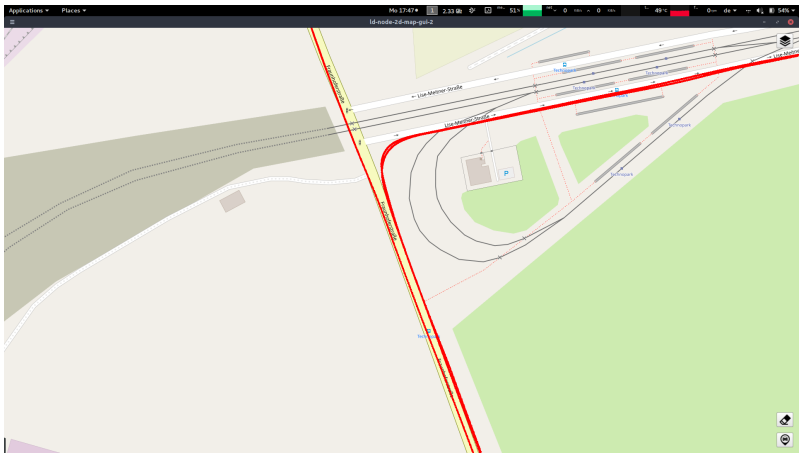
```
if(posData.solution_status == novatel::SOL_COMPUTED)
{
  GPSMeasurementT gps{};
  gps.altitude = posData.height;
  gps.latitude = posData.latitude;
  gps.longitude = posData.longitude;
  gps.timestamp =
  std::chrono::duration_cast<std::chrono::nanoseconds>
  (std::chrono::system_clock::now().time_since_epoch())
     .count();
  gps.gps_week = posData.header.gps_week;
  gps.gps_millisecs = posData.header.gps_millisecs;
  m_outputPin.push(gps, "GPSMeasurement");
```

Figure 3.1: Test vehicle

3.5 Operation

This node provides positional estimates among other things as described earlier. It can be used with another node which provides a map service to plot the current location. This can be seen in the Fig. 3.2

Figure 3.2: Operation of the node visualized using a map

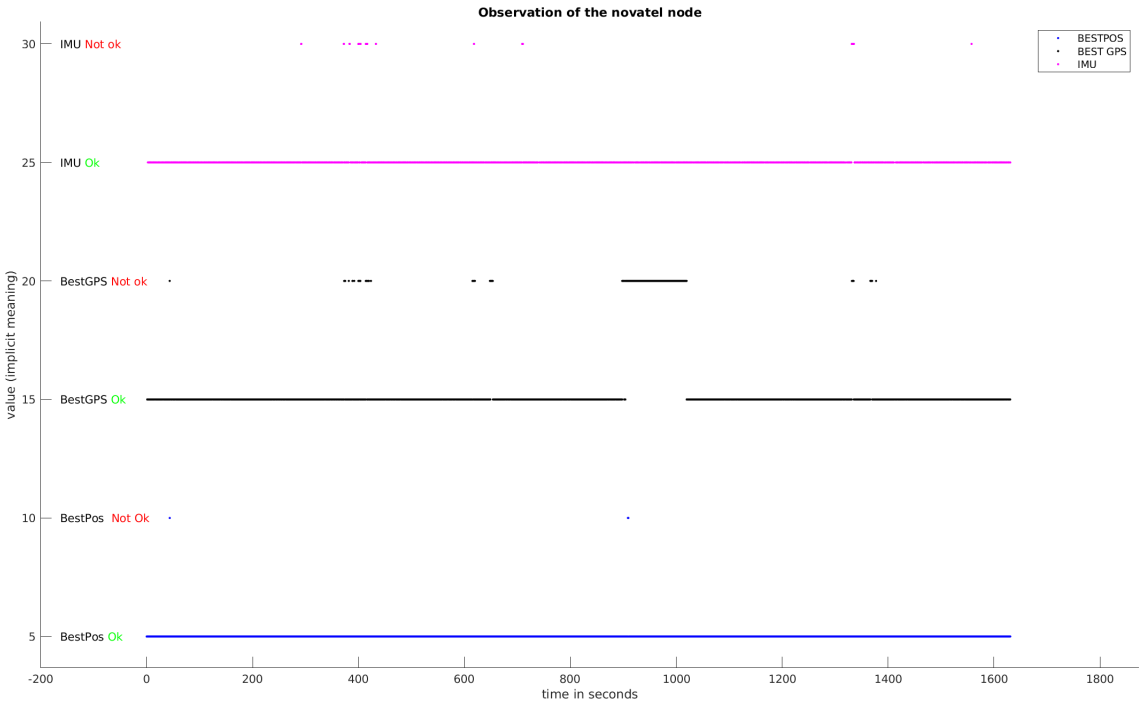


3.6 Analysis on the operation

While testing the operation of the node, it was observed that the signal from the satellites was not always reliable. occurrence of trees or tunnels for example would hamper the signal. This was of course expected. The Propak-V3 device however, also comes with an inertial measurement unit [35] which, according to design and

specifications of the vendor, should compensate for the intermittent lack of satellite signals [30]. However, it was observed that there were small but repeated durations when no position information was received from the receiver - satellite or IMU. It was important to understand exactly when this happened, i.e., I wanted to find out if there is a particular scenario which leads to this lack of signal. In addition, offsets were observed in the positional data received from the device. To do the analysis, I used the python package matplotlib [36]. The data was recorded from the node itself in csv format. In figure 3.3 you can see these messages availability is being plotted. This helped understand what was unexpected and when. BestPos is a message form the receiver which expresses the best guess of the receiver of the position [30]. IMU messages also carry coordinates, but the data is from the IMU part of the device and not the satellite GPS positional data.

Figure 3.3: Plotting the GNSS messages availability with time.



Further, to better understand what was happening, I took the GNSS receiver positional estimates and entered them into Google Maps service. The result of this can be seen in Fig. 3.4

In Fig. 3.5 you will see a zoomed out section of the map. There is a “jump” in the calculated position estimates which is clearly wrong. This was a concrete description of the problem. Why the device was generating such an offset is not known, but my speculation is that only the latitude was not getting updated correctly. The reason for this can be seen in Fig. 3.6. Note that a rotation of a few degrees leads to the positions aligning in a neat way.

Figure 3.4: Adding NovAtel positional estimates to Google Maps.

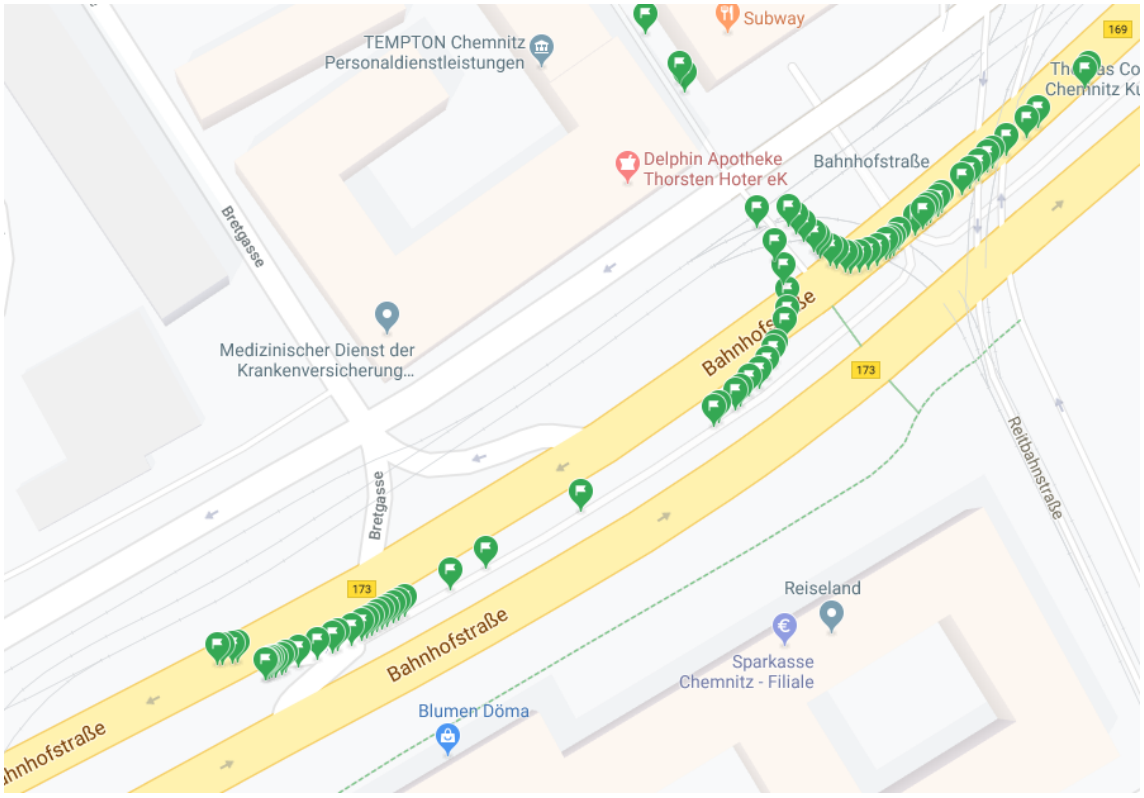


Figure 3.5: Noticing the offsets in NovAtel positional estimates on Google Maps.

Figure 3.6: Understanding the offset observed using Google Maps.

Chapter 4

Integration of machine vision cameras

4.1 Allied Vision camera integration.

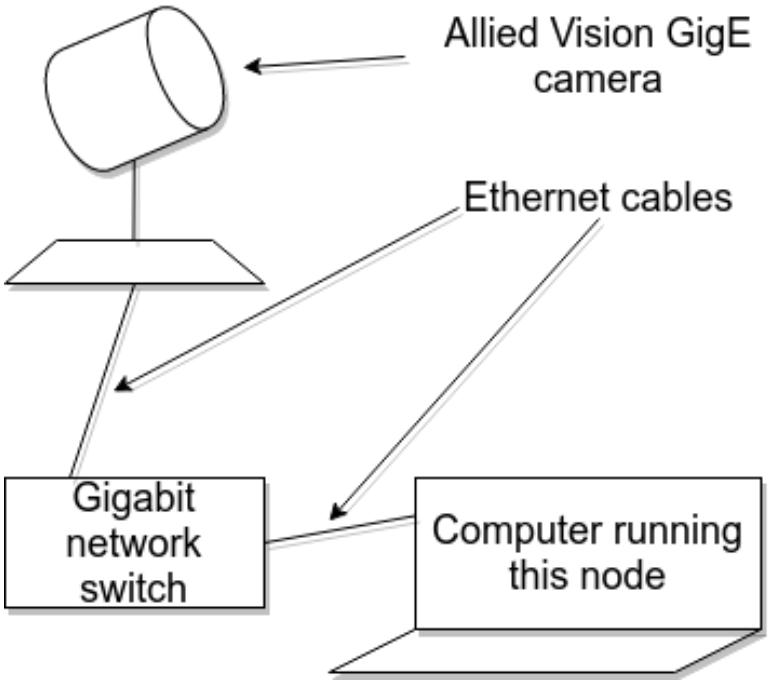
4.1.1 Motivation and Context

This node, called `ld-node-camera-avt-2` [11] is designed to be an interface between Allied Vision GigE cameras [9] and the Link2 mesh. This node allows for configuration of the cameras and then the user is able to stream images captured from the camera (with desired frame-rate, image sizes, other camera settings) to the *mesh*. Internally, this node uses the Vimba SDK provided by the manufacturer to interface with the cameras. Then it records images from the camera and writes the same thing on the mesh in *Image* format which is internally defined.

4.1.2 Hardware set-up

The camera can be Powered over Ethernet or via a Hirose connector. I have used the PoE solution. Fig. 4.1 shows what we want to achieve with the end result being that we have a network using the Ethernet on which both the camera and the computer running the node exist. We want to connect the camera to a computer which is running this node. For this to happen, it is needed to configure the IP and netmask for both camera and computer in such a way that they are on the same network. The camera was connected to a D-Link DGS-1008P Gigabit network switch [37]. The switch in turn was connected to the computer via another Ethernet cable. Once you have this hardware set-up ready, now it is time to configure your IP and netmask on both camera and the computer with the aim of bringing them on the same network so that the camera is accessible to the computer being used to run the node. This same computer will also need to run Vimba SDK [38] on it. It might be necessary to increase the MTU [39] for your Ethernet. On Linux it is done by `sudo ifconfig eth0 mtu <Desired_Rate>`. It is important to check the data transmission rate of the camera, switch, computer network card. They should all match each other or there will be failures without error messages.

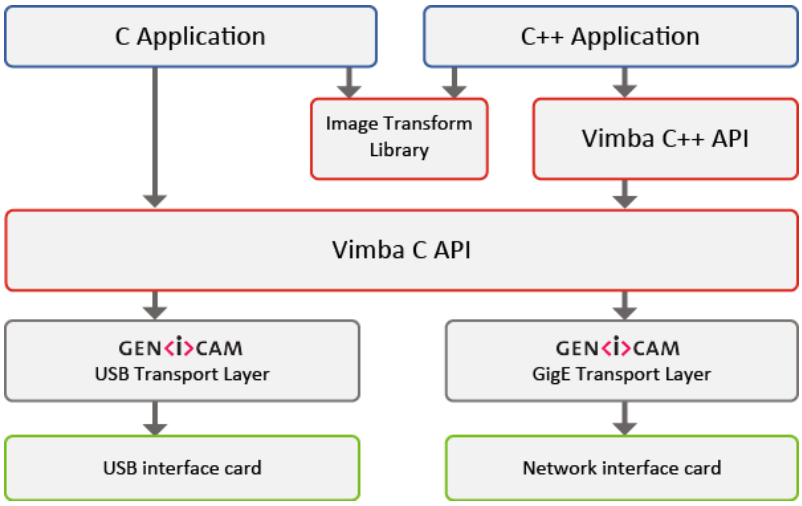
Figure 4.1: Hardware set up for GigE PoE Allied Vision camera.



4.1.3 The Allied Vision Technologies Vimba SDK and how it was used

An overview of the Allied Vision Vimba SDK can be found in Fig. 4.2. A detailed documentation of the Vimba SDK is available in [40]. A simplified view of the C++ API in particular can be seen in Fig. 4.4 [40].

Figure 4.2: Allied Vision Technologies Vimba SDK components overview [40]



I will now explain some crucial and technically interesting parts of the Vimba API. Firstly, we need to use an instance of `VimbaSystem`. "The entry point to Vimba C++ API is the VimbaSystem singleton. The VimbaSystem class allows both to control the API's behavior and to query for interfaces and cameras" [40].

```
AVT::VmbAPI::VimbaSystem& m_vimbaSystem =
AVT::VmbAPI::VimbaSystem::GetInstance();
AVT::VmbAPI::CameraPtr m_cameraPtr;
```

Next, I had to implement the interface `IFrameObserver` which contains the event handler function `FrameReceived`that gets called whenever a new frame is received [40].

```
class FrameObserver :
            public AVT::VmbAPI::IFrameObserver
{
    public:
    FrameObserver(AVT::VmbAPI::CameraPtr pCamera,
                  AVTCamDriver *pParent);

    /*!< Handler logic which runs when a frame is
       received. */
    void FrameReceived(const AVT::VmbAPI::FramePtr
       pFrame);

    private:
    /*!< Pointer to the parent camdriver class*/
    AVTCamDriver* m_parent;
};
```

To start capture of images, I had to do call the following function

```
m_camerPtr->StartContinuousImageAcquisition(NUM_FRAMES,
                        AVT::VmbAPI::IFrameObserverPtr(
new AVTCamDriver::FrameObserver(m_cameraPtr, this)));
```

And to stop acquisition I do the following (and then assuming I want to stop the node, call the shutdown function)

```
m_cameraPtr->StopContinuousImageAcquisition();
m_vimbaSystem.Shutdown();
```

Inside my implementation of *FrameReceived()* function, I write the captured image onto the *mesh*. Note the use of OpenCV [41] to convert the raw buffer of bytes to required image format.

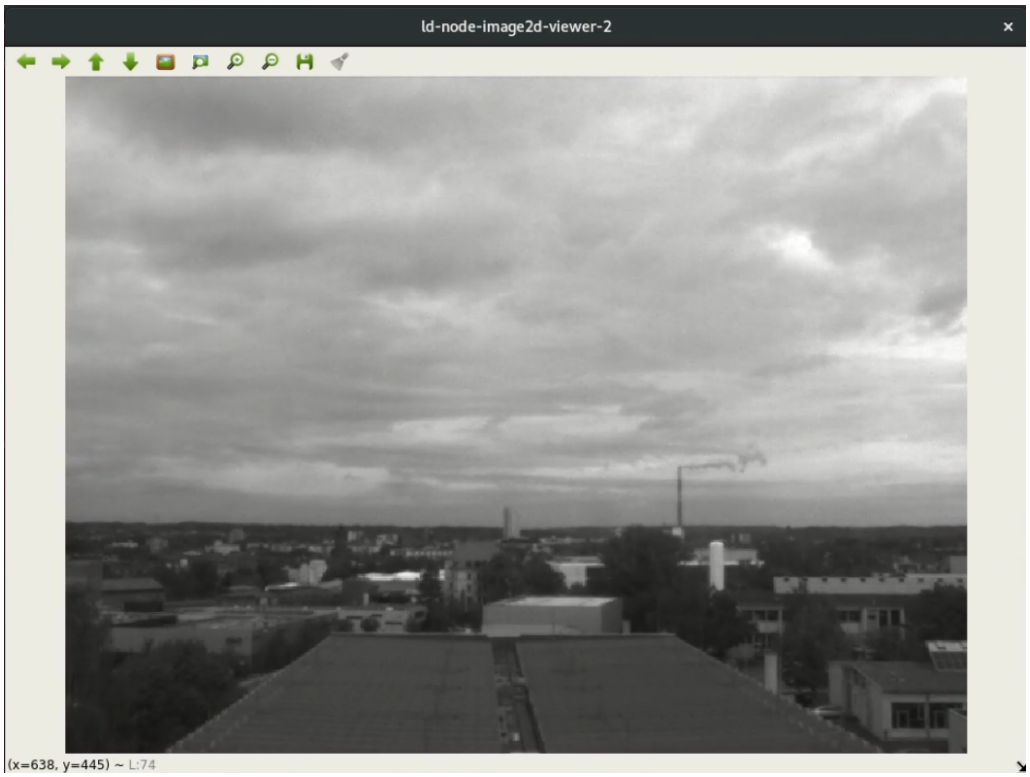
```
...
pFrame->GetBuffer(pBuffer);
cv::Mat frame(imageSize, CV_8UC1, pBuffer);
...
link_dev::ImageT currentImage
= link_dev::Interfaces::ImageFromOpenCV(frame,
              link_dev::Format::Format_GRAY_U8);

m_parent->m_outputPin.push(currentImage, "AVTCamImage")
   ;
...
```

4.1.4 Operation

When used with another *node* called `ld-node-image-viewer-2`, the images captured by this node get displayed on a GUI. A sample can be seen in Fig. 4.3.

Figure 4.3: AVT camera Node operation.



4.2 Basler camera integration

4.2.1 Motivation and Context

The node `ld-node-camera-basler` [12] is designed to be an interface between Basler GigE cameras [10] and the Link2 *mesh* [25]. This node allows for configuration of the cameras and then the user is able to stream images captured from the camera (with desired frame rate, image dimensions, other camera settings like auto exposure) to the mesh. Internally, this node uses the Pylon SDK provided by the manufacturer to interface with the cameras. Then it records images from the camera and writes the same thing on the mesh in *Image* [42] format which is internally defined using FlatBuffers [27] to store and represent images.

4.2.2 Hardware set-up

The camera used to develop this node was a BASLER acA1300-75gc [10]. The camera can be Powered over Ethernet or via a Hirose connector. In this particular setup I have used the PoE solution. To do this, a D-Link DGS-1008P Gigabit POE network switch [37] was used. A library called Pylon [21] is used internally to interface with the camera. Tools called `IpConfigurator` and `PylonViewer` are

crucial in setting up the SDK and in troubleshooting and are part of the same Pylon open source library [21].

4.2.3 The Pylon SDK and how it was used

I used the Pylon C++ API to interface with the Basler camera. An overview can be found in Fig. 4.5.
Now I will present some of the important aspects of the Pylon SDK that I used to develop this node.
I had to derive from a class called `CConfigurationEventHandler` which allows me to override a function called `OnOpened()` where I essentially out my handler code for when an image is ready to be grabbed.

```
1  class BaslerCamDriver :
2     public Pylon::CConfigurationEventHandler
3  {
4    ...
5    void OnOpened(Pylon::CInstantCamera& camera);
6    ...
```

A camera object is created using an iterator which internally iterates over all connected cameras on the same network. In our case there is only camera. This is the central object which I will use to do all the other operations.

```
1  ...
2  Pylon::DeviceInfoList_t::const_iterator it;
3  for(it = lstDevices.begin(); it != lstDevices.end();
4      ++it )
5  {
6    Pylon::CBaslerGigEInstantCamera
7    camera(tlFactory.CreateDevice(*it));
8    ...
```

After that operations to make particular settings on the camera and using it to capture images are possible. Pylon provides different "grab strategies" [21], essentially balancing on demand use and availability and providing ability to control how resource intensive the application fetching images from the camera is. The strategy I am using is called "OnByOne" [21].

```
1  camera.Open();
2  camera.StartGrabbing(Pylon::GrabStrategy_OneByOne);
```

The actual capture of images happens in a while loop like you see below. Note the use of `waitObjects` to ensure the thread with the while loop is not very CPU intensive. Essentially, the thread sleeps until there is image data ready to be grabbed. Without this mechanism there is a 4-fold increase in CPU usage. The `RetrieveResult()` function returns a pointer to the buffer inside the camera which contains the currently captured image. Note the use of OpenCV [41] library to do necessary image data manipulations. Also note the usual Link2 API to push data onto the `Mesh` on line 25 of the following listing.

```
...
while(camera.IsGrabbing() && terminate == false)
{
  if(!waitObjectsContainer.WaitForAny(0xFFFFFFFF, &
     index))
  {
    ...

    if(camera.RetrieveResult(0, ptrGrabResult,
                    Pylon::TimeoutHandling_Return))
    {
      if(ptrGrabResult->GrabSucceeded())
      {
        if(outputFormat.compare("GRAY_U8") == 0)
        {
          uint8_t *pImageBuffer =
          (uint8_t *)ptrGrabResult->GetBuffer();

          cv::Size imageSize(frameWidth, frameHeight);
          cv::Mat frame(imageSize, CV_8UC1,
             pImageBuffer);

          link_dev::ImageT currentImage =
          link_dev::Interfaces::ImageFromOpenCV(frame,
                          link_dev::Format_GRAY_U8);

          outputPin.push(currentImage,
                        "BaslerCamImage");
          ...
      ...
  ...
```

4.2.4 Operation

When used with another *node* called `ld-node-image-viewer-2` [29], the images captured by this node get displayed on a GUI. A sample can be seen in Fig. 4.6. Some manipulations on color hue and saturation were performed using the Pylon API but those changes were deemed unnecessary for the project. I analyzed the data rate generated by the camera at different settings (since different settings lead to different frame-rates and hence different requirements for bandwidth). I also created a way to measure the frames dropped during the capture process. This was useful to debug issues at run-time.

Figure 4.4: Vimba SDK simplified C++ API architecture [38].

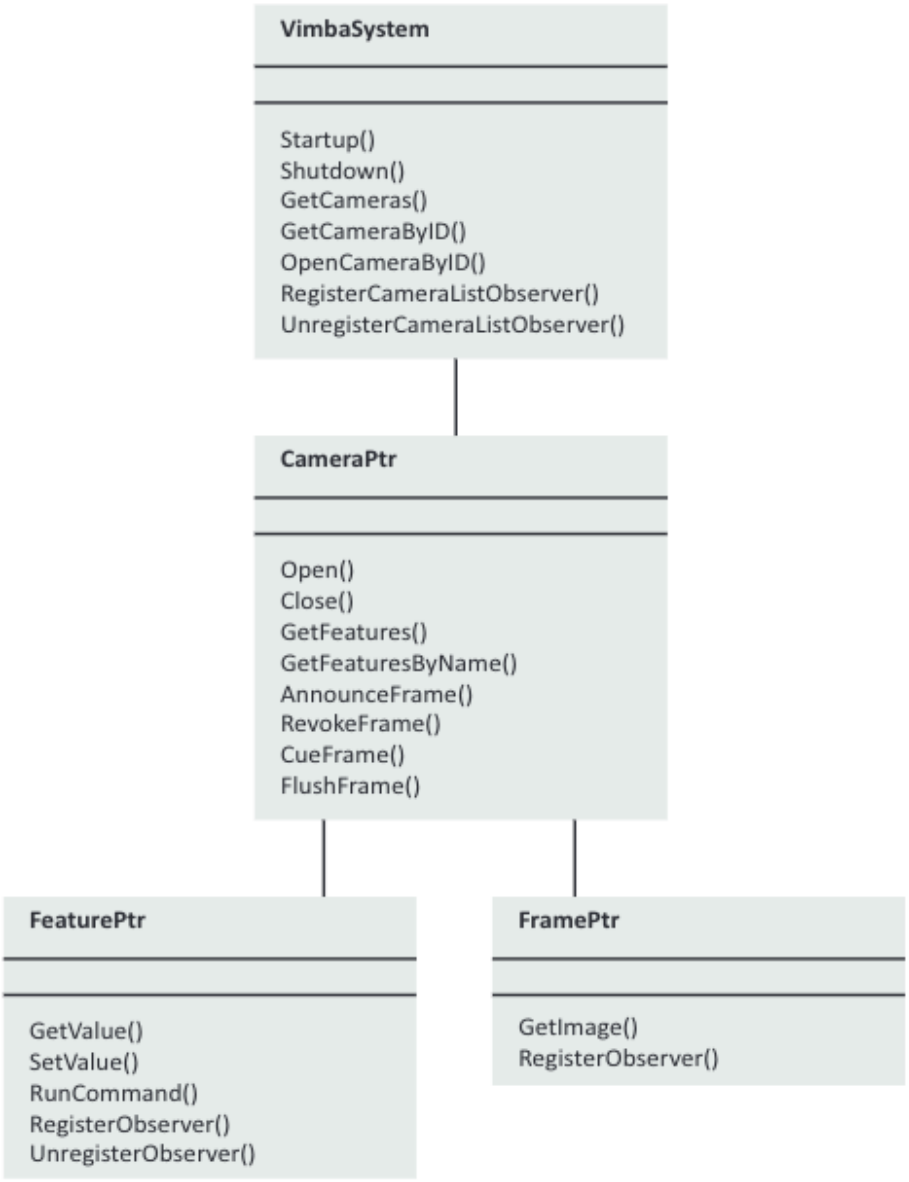


Figure 4.5: Pylon C++ API overview [21].

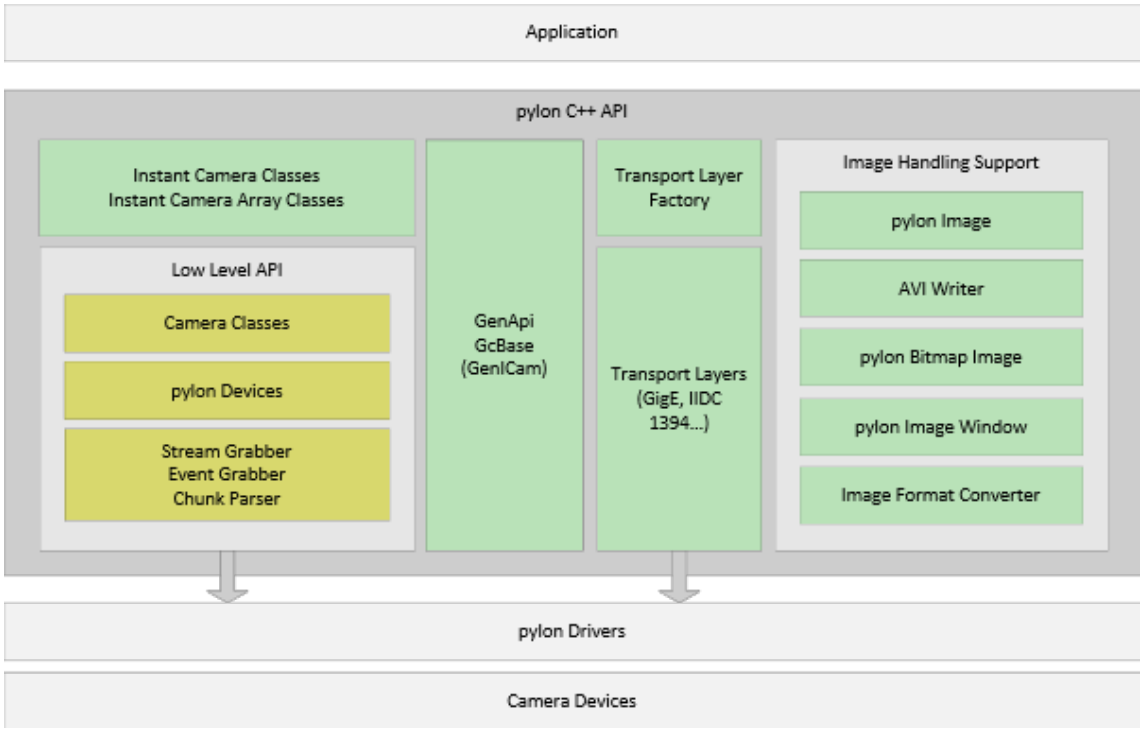
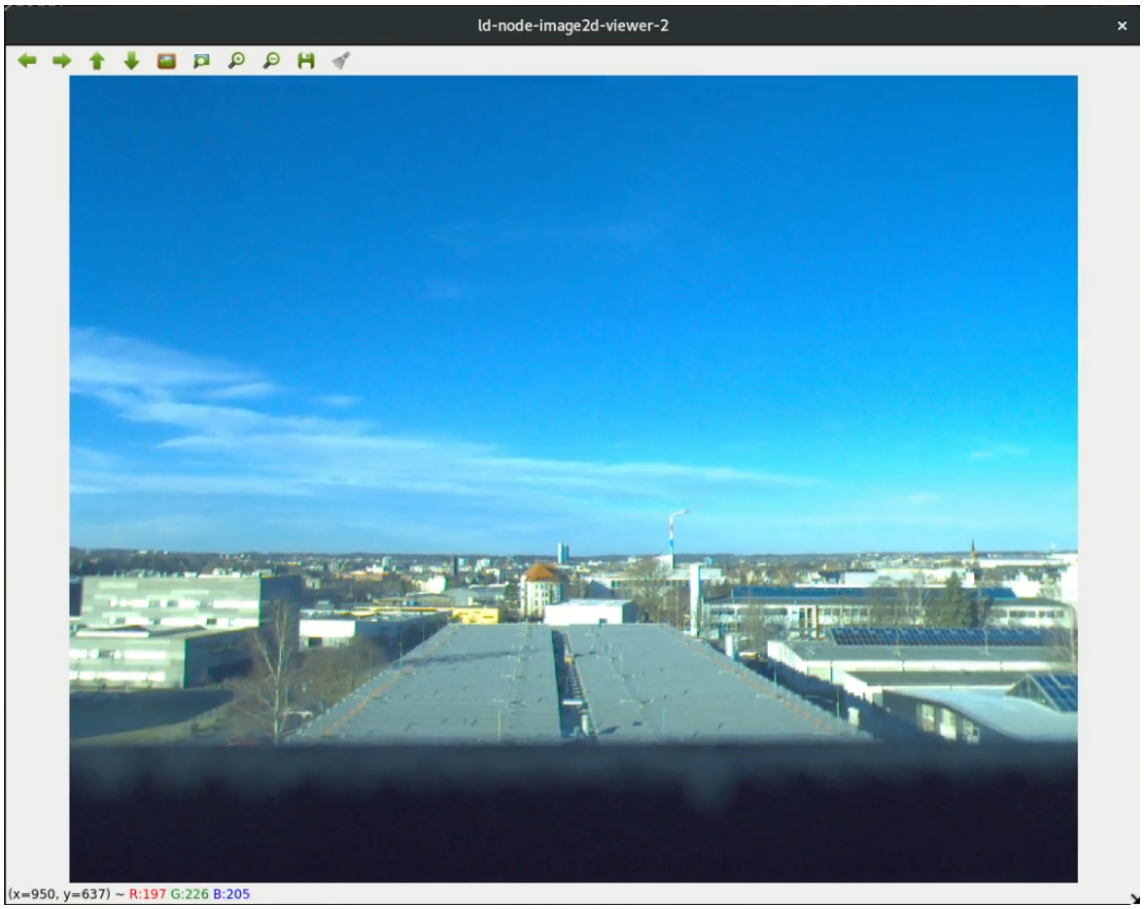


Figure 4.6: Basler camera node output with ld-node-imageviewer-2.



Chapter 5

Implementation of face detection and facial landmarks detection using the real-time Mesh

5.1 Face Detection

5.1.1 Motivation and Context

When supplied a video, this node, called `ld-node-face-detector-2` [17] detects human faces in each individual frame of the video. This node runs in 2 modes depending upon the state of the `DebugMode` configuration parameter. In normal mode this node supplies the offer which contains the image in which faces are detected and the bounding box corresponding to each face and the number of faces detected. This will be clear from the next Data Profile section. From the RFC which this node implements: "A bounding box MUST be described by the (u,v) coordinate of the top left corner and its width and height in pixels.". Note: A "bounding box" describes a subset of the image for which it is defined. Thus, in normal mode you get the original image back, with a set of set of 4 points which form rectangles in the image meant to denote faces that were found in the image. In debug mode, you get a video feed in real time (assuming a sufficiently fast computer. This is a computer with a NVIDIA Titan X GPU) with an overlay in black which shows you where the face(s) predictions were made on the frame. Internally this node uses [15] to implement the max-margin object detection algorithm [43]. It uses a pre-trained model [44] from dlib and makes heavy use of sample programs provided by dlib, viz. [44], [45] and [46].

5.1.2 Data Profile

New data formats had to be created using flatbuffers [27] in order to be able to write data on the mesh. You can see them in the following listing.

```
1  table ImageWithBoundingBoxes
2  {
3    //Image in which the bounding boxes have been defined
       .
4    imageWithFace : link_dev.Image;
```

```
5
6       //An array of bounding boxes.
7       boxes : [BoundingBox];
8
9       //Number of bounding boxes.
10      numBoundingBoxes : int;
11    }
12
13
14    table BoundingBox
15    {
16      //top left corner of the bounding box.num_filters
17      x_coordinate : int32;
18
19      //top left corner of the bounding box.
20      y_coordinate : int32;
21
22      //width of the bounding box.
23      width : int32;
24
25      //height of the bounding box.
26      height : int32;
27    }
```

5.1.3 The dlib machine learning toolkit and how it is used here

Dlib contains a wide range of machine learning algorithms. All designed to be highly modular, quick to execute, and simple to use via a clean and modern C++ API. It is used in a wide range of applications including robotics, embedded devices, mobile phones, and large high performance computing environments [13].

Subnet in dlib

From a practical standpoint, i.e. to be able to learn how to use the library, one of the first things one has to learn is the concept of a `subnet` and that of abstraction layers. It is very well described in [47]. A succinct explanation is in Fig. 5.1.

Figure 5.1: Meaning of subnet in dlib [46]

```
+-------------------------------------------------------+
| loss <-- layer1 <-- layer2 <-- ... <-- layern <-- input |
+-------------------------------------------------------+
                  ^                           ^
                  \__ subnetwork for layer1 __/
```

Building neural networks using templated aliases in dlib

Templated alias for a block which will take the sub-network passed to it and pass it through a convolution layer with `num_filters` convolutions with a 5x5 filter and 2x2 stride.

```
template <long num_filters, typename SUBNET>
using con5d =
dlib::con<num_filters, 5, 5, 2, 2, SUBNET>;
```

Templated alias for a block which will take the sub-network passed to it and pass it through a convolution layer with `num_filters` convolutions with a 5x5 filter and 1x1 stride.

```
template <long num_filters, typename SUBNET> using con5
    =
dlib::con<num_filters, 5, 5, 1, 1, SUBNET>;
```

Templated alias for a block which will take the sub-network passed to it and apply 16 convolutions in the way as defined in the first block above which is then passed to an affine layer, the entire subnet following the output of above is then subjected to another 32 convolutions of filter 5x5 and stride 2x2, followed by another affine layer and then a ReLU [48] followed by another 32 convolutions.

It applies a simple point-wise linear transformation to an input tensor. You can think of it as having two parameter tensors, A and B. If the input tensor is called $INPUT$ then the output of this layer is: $A * INPUT + B$ where all operations are performed element wise and each sample in the $INPUT$ tensor is processed separately [47].

```
template <typename SUBNET> using downsampler =
dlib::relu<dlib::affine<con5d<32, dlib::relu<dlib::
   affine<con5d<32, dlib::relu<dlib::affine<con5d<16,
   SUBNET>>>>>>>>>;
```

Templated alias for a block which will pass the sub-network passed to it through 45 convolutions of filter size 5x5 and stride 2x2 followed by an affine layer followed by a ReLU layer.

```
template <typename SUBNET> using rcon5 =
dlib::relu<dlib::affine<con5<45, SUBNET>>>;
```

Broadly speaking, there are 3 parts to a neural network definition. The loss layer, a bunch of computational layers, and then an input layer. You can see these components in the network definition below. The loss layer used is max-margin object detection [43]. You can see that an image (type dlib rgb) down-sampled with a ration of 6 to 5 forms the input. The middle layers define the computation the network will do to transform the input into whatever we want. Here we see 4 layers of the block defined just above being used.

```
using neuralNetType =
dlib::loss_mmod<dlib::con<1, 9, 9, 1, 1,
rcon5<rcon5<rcon5<downsampler<
dlib::input_rgb_image_pyramid<dlib::pyramid_down<6
   >>>>>>>>;
```

The above is the final neural network that is used. Note that this *design* of neural network is taken from [43].

Next we convert our images from video input to dlib using the following call:

```
dlib::assign_image(dlibImage, dlib::cv_image<dlib::
    bgr_pixel>(currentFrame));
```

Now tell the face detector to give us a list of bounding boxes around all the faces it can find in the image.

```
std::vector<dlib::mmod_rect>  detFaces = m_neuralNet(
    dlibImage);
```

5.1.4 Fetching data from a different node

Note that in this node we also have an *Input pin*. That is how we get the images on which we run the algorithm described in the previous subsection. Following listing makes clear how that works. Note the callback `DetectFaces()` in which I put the code for handling a new incoming image frame.

```
m_inputPin.addOnDataCallback("l2demand:/
    videoinput_thruimages",
[&](const link_dev::ImageT& receivedImage)
{
     DetectFaces(link_dev::Interfaces::ImageToOpenCV(
        receivedImage));
});
```

5.1.5 Pushing the data on the mesh

For each face that is found in current frame, create a `BoundingBoxT` object and add a pointer to this object to the vector of bounding boxes in the flatbuffer [27].

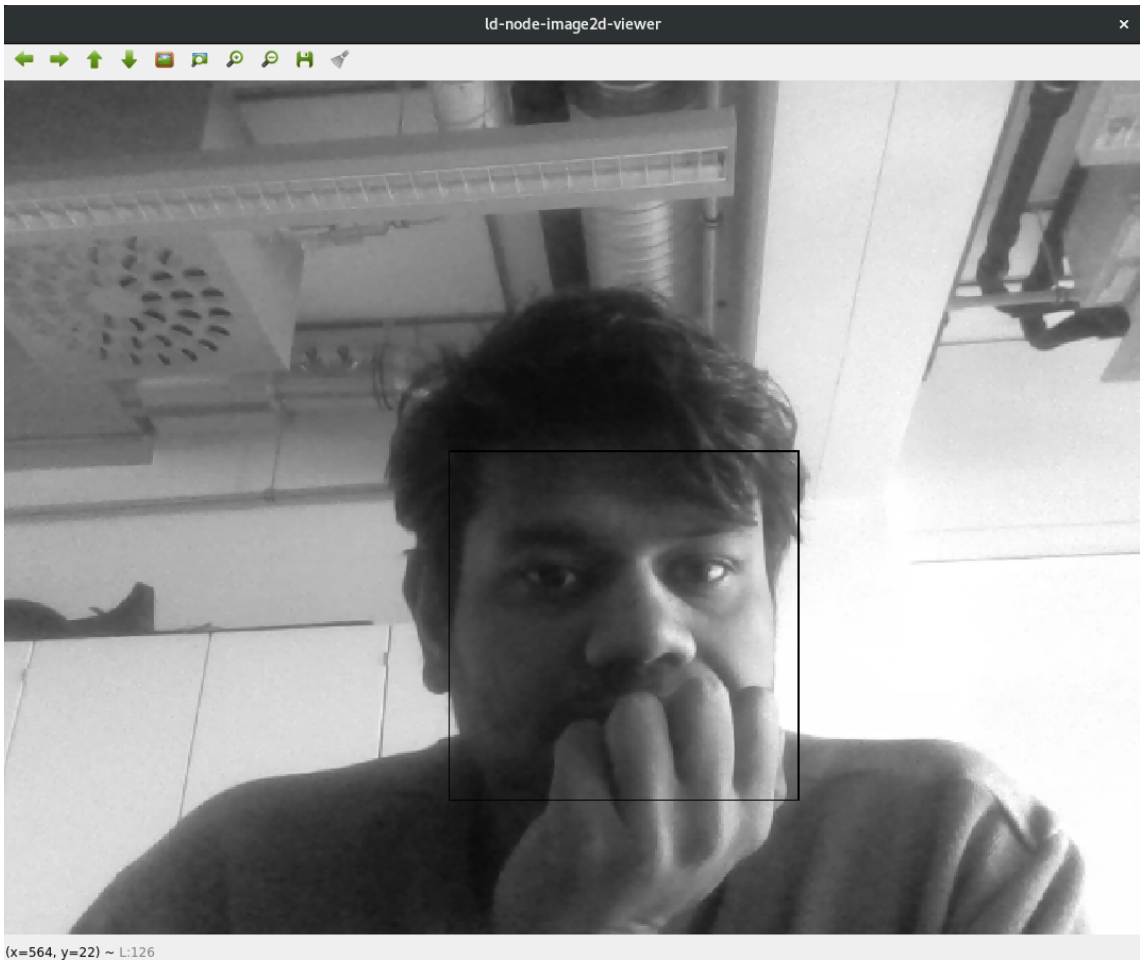
```
BoundingBoxT currentBB;
currentBB.x_coordinate = face.rect.left();
currentBB.y_coordinate = face.rect.top();
currentBB.width = face.rect.right()-face.rect.left();
currentBB.height = face.rect.bottom()-face.rect.top();

imageAndBB.boxes.push_back(std::make_unique<
    BoundingBoxT>(currentBB));
```

5.1.6 Operation

When used in Debug mode with `ld-node-imageviewer-2` [29], the output in Fig. 5.2 is produced. Note the demarcation of face. The camera used in the operation was an *Allied Vision Mako GigE camera* [9] using `ld-node-camera-avt-2` [11] node.

Figure 5.2: Debug Output from ld-node-facedetector-2



5.2 Facial Landmarks Detection

5.2.1 Motivation and Context

The *node* `ld-node-facial-landmark-detector-2` [18] uses TensorFlow library and pre-built models to detect 68 "face key points" on the image of face supplied. This image of face is generated by `ld-node-face-detector-2` [17] as described in the previous section. Note that `ld-node-face-detector-2` writes output in the form of the below flatbuffer [27] listing.

```
1  //Contains an Image and one or more set(s) of
2  //co-ordinates that define a bounding box
3  //inside the image.
4
5  table ImageWithBoundingBoxes
6  {
7    //Image in which the bounding boxes have been defined
8    imageWithFace : link_dev.Image;
9
10   //An array of bounding boxes.
```

```
    boxes : [BoundingBox];

    //Number of bounding boxes.
    numBoundingBoxes : int;
}
```

The video output from a camera can be used to supply an input stream of images and every frame of the video will be scanned for faces (by `ld-node-face-detector-2`) and subsequently facial landmarks can be also marked by this node in real time (assuming a processor and GPU fast enough to do so). The pre-built TensorFlow models used in this node are obtained from [16] and [49].

5.2.2 TensorFlow C++ API and how it is used here

One of the first things that is done is to load the pre-built graph [50] and create a session [51] with it.

```
1  ...
2  tensorflow::GraphDef graph_def;
3
4  /*Read the protobuf graph supplied by user*/
5  Status load_graph_status =
6  ReadBinaryProto(tensorflow::Env::Default(),
7                  graph_filename, &graph_def);
8  ...
9
10 /*Create a tensorflow::session to execute the above
      graph*/
11 session.reset(tensorflow::NewSession(
12 tensorflow::SessionOptions()));
13
14 Status session_create_status =
15 session->Create(graph_def);
16 ...
```

Later, when individual frames and the bounding boxes that represent faces in the image are processed one by one by the node, we create an input tensor using the bounding box from the input and then use a session to make the predictions on line 12 in the below listing.

```
1  ...
2  cv::Mat norm_face;
3  croppedFace.convertTo(norm_face,
4                        CV_32FC3, 1.0 / 255, 0);
5  Tensor inpTensor(DT_FLOAT,
6          TensorShape({ 1,256,256,3 }));
7  float * inpPt = inpTensor.flat<float>().data();
8  cv::Mat inpImg(256, 256, CV_32FC3, inpPt);
9  norm_face.convertTo(inpImg, CV_32FC3);
10 ...
```

```
11
12  Status run_status = session->Run({ { input_layer,
13                                       input_tensor } },
14                                     { output_layer },
15                                                   {},
16                                    &output_tensors);
17  ...
```

5.2.3 Fetching Images with Bounding Boxes from ld-node-face-detector-2 and publishing output

Note that we need `ImageWithBoundingBoxesT` in order to function. This *subscription* is done in the following way -

```
1  m_inputPin.addOnDataCallback("l2demand:/
      image_with_bounding_boxes",
2  [&](const ImageWithBoundingBoxesT& imageWithBB)
3  {
4      HandleNewFrame(imageWithBB.imageWithFace,
          imageWithBB.boxes);
5  });
```

Some image transformations are used after running the TensorFlow session to get the landmarks from the predicted output and these are published onto the mesh.

```
1   m_outputPin.push(
2       link_dev::Interfaces::ImageFromOpenCV(
3                       allFacesLandmarks,
4       link_dev::Format::Format_GRAY_U8),
5               "l2offer:/imagesWithLandmarks");
```

5.2.4 Operation

I created a *mesh* with the following *nodes*

- `ld-node-image2d-viewer-2` [29]
- `ld-node-face-detector-2` [17]
- `ld-node-webcam-2` [52]
- `ld-node-facial-landmark-detector-2` [18]

This can be visualized with another node called `ld-node-mesh-viewer` in Fig. 5.4. The output of this node can also be visualized in debug output and this can be seen in Fig. 5.3

Figure 5.3: Debug output from ld-node-facial-landmark-detector-2

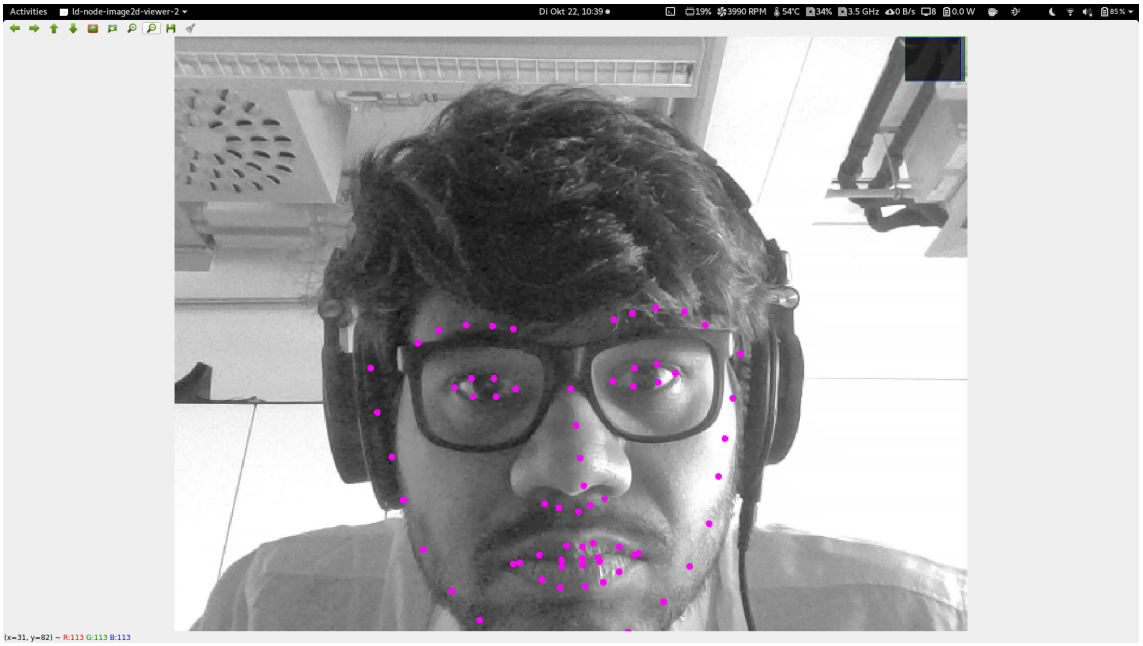


Figure 5.4: View of the mesh for facial landmark detection

ld-node-mesh-viewer (3484): link_dev

link_dev Stop

ID	Data Rate	Total Data	DataObject Rate	Total DataObjects
▼ ld-facial-landmark-detector-2				
▼ imagesWithLandmarks				
l2offer:/imagesWithLandmarks	899,9 kB/s	6,15 MB	1.00 Hz	7
▼ imagesWithBoundingBox				
l2demand:/image_with_bounding_boxes	0 B/s	2,05 MB	0.00 Hz	7
▼ sample-ld-node-image2d-viewer-2				
▼ image-input				
l2demand:/img-messages	0 B/s	5,27 MB	0.00 Hz	6
▼ ld-node-facedetector-2				
▼ video_output				
l2offer:/debug_video_output	0 B/s	0 B	0.00 Hz	0
l2offer:/image_with_bounding_boxes	300,1 kB/s	4,69 MB	1.00 Hz	16
▼ video_input				
l2demand:/videoinput_thruimages	300,0 kB/s	24,61 MB	1.00 Hz	84
▼ ld-node-webcam-2				
▼ webcam-output				
default	300,0 kB/s	24,61 MB	1.00 Hz	84
▼ mesh-viewer-3484				
▼ metrics-input				
l2demand:/metrics	2,8 kB/s	38,9 kB	7.00 Hz	97

Chapter 6

Package configuration using conda

In order to maintain a repository of external libraries and packages, `conda` [19] can be used. Some important steps that I had to follow to do package configuration for the Pylon SDK [21] with Gitlab CI pipeline [20] are described in this chapter. The end result was a *package-config* [53] for the Basler Pylon SDK which can be deployed with conda [19] and can be found in a Github repository [22].

6.1 The meta.yaml file

This file is necessary for things like specifying requirements for the build and host environments and specifying where exactly the package download is available from.

```
1  ...
2  package:
3  name: "pylonsdk"
4  version: "5.2.0"
5
6  source:
7  - url: https://hydrogen.draive.com/files/link_dev/pylon
       -5.2.0.13457-x86_64.tar.gz # [linux and x86_64]
8  - url: https://hydrogen.draive.com/files/link_dev/pylon
       -5.2.0.13457-arm64.tar.gz # [linux and aarch64]
9  - url: https://hydrogen.draive.com/files/link_dev/pylon
       -5-20191104T161031Z-001.zip # [win]
10
11 build:
12 number: {{ BUILD_VERSION }}
13
14 requirements:
15 build:
16 - {{ compiler("c") }}
17 - {{ compiler("cxx") }}
18
19 ...
```

6.2 Telling conda how to create the necessary folder structure

This can be done like this -

```
if [[ $target_platform == 'linux-aarch64' ]]; then
tar -xf pylonSDK-5.2.0.13457-arm64.tar.gz
elif [[ $target_platform == 'linux-64' ]]; then
tar -xf pylonSDK-5.2.0.13457-x86_64.tar.gz
fi

PylonSDK_LIB_TARGET=$PREFIX/lib/
PylonSDK_INCLUDE_TARGET=$PREFIX/include/

mkdir -p $PylonSDK_LIB_TARGET
cp -r pylon5/lib64/* $PylonSDK_LIB_TARGET


mkdir -p $PylonSDK_INCLUDE_TARGET
cp -r pylon5/include/* $PylonSDK_INCLUDE_TARGET

cp $RECIPE_DIR/prefix/* $PREFIX -r

echo "true";
```

6.3 Writing a CMake target import file for required configuration

Essentially we are letting `find_package` [54] find our manually created package. This consists of two major steps - creating imported targets [55] and setting property [56]. You can see this happening in the below listing.

```

...

# Create imported targets
add_library(PylonGCBaseLibrary SHARED IMPORTED)
set_target_properties(PylonGCBaseLibrary PROPERTIES
   INTERFACE_INCLUDE_DIRECTORIES
"${_IMPORT_PREFIX}/include")

add_library(PylonGenAPILibrary SHARED IMPORTED)
set_target_properties(PylonGenAPILibrary PROPERTIES
   INTERFACE_INCLUDE_DIRECTORIES
"${_IMPORT_PREFIX}/include")

add_library(PylonBaseLibrary SHARED IMPORTED)
```

```
14  set_target_properties(PylonBaseLibrary PROPERTIES
       INTERFACE_INCLUDE_DIRECTORIES
15  "${_IMPORT_PREFIX}/include")
16
17  add_library(PylonUtilityLibrary SHARED IMPORTED)
18  set_target_properties(PylonUtilityLibrary PROPERTIES
       INTERFACE_INCLUDE_DIRECTORIES
19  "${_IMPORT_PREFIX}/include")
20
21
22  ...
23  # Import targets for configuration "Release"
24  set_property(TARGET PylonGenAPILibrary APPEND PROPERTY
25  IMPORTED_CONFIGURATIONS RELEASE)
26  set_property(TARGET PylonGCBaseLibrary APPEND PROPERTY
27  IMPORTED_CONFIGURATIONS RELEASE)
28  set_property(TARGET PylonUtilityLibrary APPEND PROPERTY
29  IMPORTED_CONFIGURATIONS RELEASE)
30  set_property(TARGET PylonBaseLibrary APPEND PROPERTY
31  IMPORTED_CONFIGURATIONS RELEASE)
32
33  ...
34
35  set_target_properties(PylonGenAPILibrary PROPERTIES
       IMPORTED_LOCATION_RELEASE
36  "${_IMPORT_PREFIX}/lib/libGenApi_gcc_v3_1_Basler_pylon.
       so")
37
38  set_target_properties(PylonUtilityLibrary PROPERTIES
       IMPORTED_LOCATION_RELEASE
39  "${_IMPORT_PREFIX}/lib/libpylonutility-5.2.0.so")
40
41  set_target_properties(PylonBaseLibrary PROPERTIES
       IMPORTED_LOCATION_RELEASE
42  "${_IMPORT_PREFIX}/lib/libpylonbase-5.2.0.so")
43
44  set_target_properties(PylonGCBaseLibrary PROPERTIES
       IMPORTED_LOCATION_RELEASE
45  "${_IMPORT_PREFIX}/lib/libGCBase_gcc_v3_1_Basler_pylon.
       so")
46
47  ...
```

6.4 Writing the CMakeLists file

Once the steps in the above sections are completed, the `CMakeLists.txt` from the below listing will work for our manually created package.

```
...
find_package(pylonsdk REQUIRED)
add_executable(pylon_test main.cpp)
target_include_directories(pylon_test PUBLIC)
target_link_libraries(pylon_test
                  PUBLIC PylonGenAPILibrary
                  PUBLIC PylonBaseLibrary
                  PUBLIC PylonGCBaseLibrary
                  PUBLIC PylonUtilityLibrary)
```

6.5 Writing a CI Test for the newly created package

A small test was created like in the below listing

```
#include <iostream>
#include <GenICamVersion.h>
#include <pylon/PylonIncludes.h>


int main()
{
    Pylon::PylonInitialize();
    Pylon::PylonTerminate();
    return 0;
}
```

The above test is executed by the CI pipeline using the code in the following listing

```
echo "Run simple test program (makes initialize and
   terminate API calls)"

mkdir -p pylon_test/build && cd pylon_test/build
cmake -DCMAKE_CXX_COMPILER=$GXX -DCMAKE_CXX_FLAGS=
   $CXXFLAGS ..
make -j$CPU_COUNT pylon_test
./pylon_test
```

Chapter 7

Conclusion

A wide variety of problems which include autonomous driving are solved using a mix of different kinds of sensors like GNSS, camera, Radar, LiDAR. Independent components of a system might have the need for particular type of sensor data while there might be a lot more of sensor data being collected in the overall system. Different components might need the same sensor data, i.e. a many to one relationship. Most importantly, we are generally dealing with problems which need the data in real-time. This creates the need of a sensor fusion platform like Link2 [5].

Reverse engineering using simple tools like [3] can reveal a lot about product design and even operation as can be seen from the work done for hacking the GoPro Hero7 [1] and its Smart Remote [2] accessory in [4].

It is not enough to simply follow the API provided by the vendor of the sensor in order to guarantee performance. A wide variety of tests should be created to test the sensor in different conditions. Benchmark should be created for defining adequate performance and the sensor should be tested in all such circumstances. Software should assume failure can occur and should provide a way to register the failure for further analysis.

Sensors like cameras have an extremely broad range of capabilities. Selecting the right model for the right job is often not trivial, especially when cost is accounted for. Different vendor API's will have different properties and behavior and relying too much on the "vendors way of thinking" is not a good idea.

Machine Learning is a fast evolving field with technologies changing fast. Learning to be an effective applications engineer in this field is an important skill to develop. This mainly consists of familiarizing oneself with different tool-kits and frameworks to do machine learning. Training models is a hard job. Not only does it require a lot of very specific technical and mathematical skills, but also considerable computing power resources and access to cutting edge and expensive hardware like Graphic Cards. However, by clever selection of models which other people have already trained, we can get a lot done in terms of proof of concept, selecting the right machine learning technique and experimentation. Such experimentation goes a long way in fulfilling scientific demands which are otherwise too resource consuming to be done. In order to keep up, a constant state of experimentation and an attitude of quick prototyping is necessary. This would usually mean keeping up with how people organize raw data in different machine learning frameworks and toolkits like TensorFlow, PyTorch, dlib, many others and learning about novel machine learning models that are published and understanding how to use them for our own

applications.

Effective documentation of software is necessary to provide a smooth transition from software proof of concepts to final delivery of software products. I achieved this from writing RFC's to simple README files for every software component developed.

Package management [19] goes a long way to develop effective and hassle free open source software. Continuous Integration and Continuous Delivery using mature pipelines [20] offered by GitLab for example, are very effective tools to manage software repositories and keeping the product *shippable* at all times.

I also found that avoiding unnecessary jargon, keeping clarity of thought and process, understanding basic and essential building blocks of any system or framework, using uncomplicated language both in documentation and while writing code; are unparalleled tools to produce effective software systems.

Appendix A

Guidelines for conducting effective test drives

A.1 The guidelines

I had to conduct a total of 11 test-drives when I was working on the GNSS receiver. I enumerated a set of points which are meant as guidelines for anyone who is going for a test-drive. These guidelines were added to the user-guide [23] and I am repeating them here:

- Write down why you are going for this test drive. What do you aim to accomplish? What do others expect? Sure you have a good idea why, but have you confirmed with others interested in this test drive? Talk with them. Try to frame a simple single line "purpose" and confirm this with others. This is the most important thing to consider. It may affect the way you conduct the test drive, the locations you choose to go, various settings, configurations.

- Once you have enlisted the solid reason(s), motivation(s) for the test drive, think what data you would generate and in what format. Have a clear picture of what data you want to record and how you will do it. Do you need extra tools? Is your recording fully automatic or requires manual intervention? Do you have any concerns about your tools not working? Do you need to test them first before you start the actual test drive? Think about any likely but easily unforeseen situations. For example, if you are using some sort of map service and it is possible that you might not have Internet during the test drive, then you would probably want to cache the map before you start! Also, many times the data that you generate during your test drive is going to be volume intensive. So try to record data (for example your log files) in a way that it is easy to read and parse it using programming.

- Do you have any special requirements in the test drive? Should the car go through tunnels? Does the car need to be on a busy street or a highway? If there are such special requirements, make sure you talk about it with the person driving the car beforehand. Ideally, you should have a route prepared already.

- Might be beneficial to create small tables or text files in which you type in various kinds of details during the test drive. You should keep a "diary style"

or "minutes of meeting" style record of the things that happen during the test drive. This could come in very handy.

- To sum it up shortly, be ready with a detailed "test drive plan" and have someone look at it before the test drive. Communicate with your driver about your objectives from the test drive and try to gather as much information as possible and try to make it easier for you do so (better organization, automation, tools).

A.2 Example of a test drive plan

Warm up and sanity test Start node while car is stationary but out in the open. Test if the node works. Cache the map.

Check operation of node without IMU Start recorder. Start map-gui. Don't enable IMU. Test if node works. Check if logs are created properly. Stop recorder. Save the recorded file. Stop map-gui.

Check operation of node with IMU Start recorder. Start map-gui. Enable IMU. Test if node works. Check if logs are created properly. Stop recorder. Save the recorded file. Stop map-gui.

Check if "status" is getting recorded when solution status is not good Start recorder. Start map-gui. Enable IMU. Cover gnss receiver with hands. Test if node works. Check if logs are created properly. Stop recorder. Save the recorded file. Stop map-gui.

A.3 Example of a test drive log

16:38 Trees overhead

16:39 A lot of IMU outage

16:39:48 Trees

16:41 Small under-bridge

16:46:32 Big tunnel

16:48 Possible offset or missing BESTPOS messages

16:51 GPS and INS both failed - trees and bad roads

17:24:31 Trees

17:26:25 Tunnel (big)

17:28:30 Building and some trees

17:33 Small under-bridge

Appendix B

Guidelines for using media in software documentation

I wrote a few guidelines for using media like pictures, GIF's, video in software documentation which was later included in the user-guide [23]. Here are the points I came up with:

- Media is a great way to add substantial value to your documentation, making it very easy to provide instant context to what problem your software solves, or how. A short demonstration can also be generally included in the README via pictures, videos, GIF's. Below are some guidelines which can help you to add better value minus the clutter.
- Your media should be on point. It should not contain anything else that may confuse the reader. For example, if you want to show a camera, then having a laptop in the same picture as the camera may lead the reader to believe that perhaps the laptop is also needed for the operation of the camera.
- Environment of the media should be consistent with the purpose of the project. For example, if your project is about using computer vision on vehicles, then your demonstration should use vehicles (and not apples for example).
- When showing a demonstration, it would be nice to have a textual prompt somewhere on the screen which "narrates" the demonstration to the reader. Note that GIF's don't have sound.
- Sometimes you may be tempted to use a picture where a simple diagram might suffice. Don't forget to use the "simplest way" first. Don't use a picture (or other media) where a diagram would be enough.
- Always aim for easier, faster and better understanding of the reader when you embed media to your software documentation.
- The media that you use, should in no way divulge any personal information or any identification of instruments, tools, infrastructure in any way. Do not overlook privacy concerns of not just yourself, but others as well.

Appendix C

Git forking workflow

Throughout my tenure of internship, I was required to follow the "forking model" git workflow to organize my source code and for activities like continuous integration, code review, collaboration with the rest of the team. Detailed information on this subject can be found at [57] and [58].

In short, the git forking workflow is to be followed through the following steps:

- Fork a Git repository.
- Clone the forked repository to your local system.
- Add a Git remote for the original repository.
- Create a feature branch in which to place your changes.
- Make your changes to the new branch.
- Commit the changes to the branch.
- Push the branch to Git.
- Open a pull request from the new branch to the original repo.
- Clean up after your pull request is merged.

I found this style of managing source code change, versioning and collaboration to be effective for the following reasons:

- It uses git! Git is very popular and implicitly supports the decentralized way of building software.
- The model is rather simple.
- It is easy to get used to the model and not make a lot of mistakes.
- It establishes a strict contract for a developer in terms of making contributions.

Bibliography

[1] (Feb. 12, 2020). Gopro hero7 black — insanely smooth 4k video, [Online]. Available: https://gopro.com/en/us/shop/cameras/hero7-black/CHDHX-701-master.html (visited on 02/12/2020).

[2] (Feb. 12, 2020). Gopro smart remote control, [Online]. Available: https://gopro.com/en/us/shop/accessories/smart-remote/ARMTE-002.html (visited on 02/12/2020).

[3] (Feb. 12, 2020). Aircrack-ng, [Online]. Available: https://www.aircrack-ng.org/ (visited on 02/12/2020).

[4] (Feb. 20, 2020). Numericaladvantage/goprohack: Hacking the wireless udp communication between gopro hero 7 and smart remote device., [Online]. Available: https://github.com/NumericalAdvantage/GoProHack (visited on 02/20/2020).

[5] (Dec. 9, 2019). Reactive mesh for realtime data processing – draive.link documentation, [Online]. Available: https://draive.com/docs/link2/?cpp#reactive-mesh-for-realtime-data-processing (visited on 02/11/2020).

[6] (Mar. 8, 2020). What is gnss? — european global navigation satellite systems agency, [Online]. Available: https://www.gsa.europa.eu/european-gnss/what-gnss (visited on 03/08/2020).

[7] (Feb. 12, 2020). Propak-v3^{TM} triple-frequency gnss receiver — high precision gnss and gps receivers — novatel gnss receiver — novatel, [Online]. Available: https://www.novatel.com/products/gnss-receivers/enclosures/propak-v3/ (visited on 02/12/2020).

[8] (Feb. 20, 2020). Numericaladvantage/ld-node-novatel-gnss-ins-2: A link2 node to interface with the novatel propak-v3 gnss/imu, [Online]. Available: https://github.com/NumericalAdvantage/ld-node-novatel-gnss-ins-2 (visited on 02/20/2020).

[9] (Feb. 12, 2020). Mako g documentation - allied vision, [Online]. Available: https://www.alliedvision.com/en/support/technical-documentation/mako-g-documentation.html (visited on 02/12/2020).

[10] B. AG. (Feb. 12, 2020). Basler ace aca1300-75gc - area scan camera, [Online]. Available: https://www.baslerweb.com/en/products/cameras/area-scan-cameras/ace/aca1300-75gc/#tab=features (visited on 02/12/2020).

[11] (Feb. 20, 2020). Numericaladvantage/ld-node-camera-avt-2: A link2 node to interface with allied vision mako g monochrome machine vision camera., [Online]. Available: https://github.com/NumericalAdvantage/ld-node-camera-avt-2 (visited on 02/20/2020).

[12] (Feb. 20, 2020). Numericaladvantage/ld-node-camera-basler: A link2 node to interface a basler ace series machine vision camera., [Online]. Available: https://github.com/NumericalAdvantage/ld-node-camera-basler (visited on 02/20/2020).

[13] D. E. King, "Dlib-ml: A machine learning toolkit", *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[14] (Feb. 10, 2020). Tensorflow c++ reference — tensorflow core v2.1.0, [Online]. Available: https://www.tensorflow.org/api_docs/cc (visited on 02/18/2020).

[15] (Feb. 12, 2020). Dlib c++ library, [Online]. Available: http://dlib.net/ (visited on 02/12/2020).

[16] Y. Feng, F. Wu, X. Shao, Y. Wang, and X. Zhou, "Joint 3d face reconstruction and dense alignment with position map regression network", in *ECCV*, 2018.

[17] (Feb. 20, 2020). Numericaladvantage/ld-node-facedetector-2: A link2 node to detect faces in a stream of video., [Online]. Available: https://github.com/NumericalAdvantage/ld-node-facedetector-2 (visited on 02/20/2020).

[18] (Feb. 20, 2020). Numericaladvantage/ld-node-facial-landmark-detector-2: A link2 node to detect facial features, [Online]. Available: https://github.com/NumericalAdvantage/ld-node-facial-landmark-detector-2 (visited on 02/20/2020).

[19] (Oct. 31, 2019). Conda — conda documentation, [Online]. Available: https://docs.conda.io/en/latest/ (visited on 02/18/2020).

[20] (Feb. 18, 2020). Creating and using ci/cd pipelines — gitlab, [Online]. Available: https://docs.gitlab.com/ee/ci/pipelines.html (visited on 02/18/2020).

[21] B. AG. (Feb. 14, 2020). Pylon open source, [Online]. Available: https://www.baslerweb.com/en/products/software/basler-pylon-camera-software-suite/pylon-open-source-projects/ (visited on 02/17/2020).

[22] (Feb. 20, 2020). Numericaladvantage/pylonsdk: A package config for the open source pylon sdk form baser ag, [Online]. Available: https://github.com/NumericalAdvantage/pylonsdk (visited on 02/20/2020).

[23] (Jan. 20, 2020). Introduction · student guide, [Online]. Available: http://pc-nobi.etit.tu-chemnitz.de/userguide/ (visited on 02/12/2020).

[24] (Jan. 16, 2020). Introduction - link developers rfc, [Online]. Available: https://draive.com/link_dev/rfc/ (visited on 02/14/2020).

[25] (Dec. 9, 2019). Core components – draive.link documentation, [Online]. Available: https://draive.com/docs/link2/?cpp#core-components (visited on 02/14/2020).

[26] (Dec. 9, 2019). Pins – draive.link documentation, [Online]. Available: https://draive.com/docs/link2/?cpp#pins (visited on 02/14/2020).

[27] (Jun. 20, 2019). Flatbuffers: Flatbuffers, [Online]. Available: https://google.github.io/flatbuffers/ (visited on 02/14/2020).

[28] (Dec. 9, 2019). Subscription and mapping – draive.link documentation, [Online]. Available: https://draive.com/docs/link2/?cpp#subscription-and-mapping (visited on 02/14/2020).

[29] (Jan. 16, 2020). Image2dviewer - link developers rfc, [Online]. Available: https://draive.com/link_dev/rfc/005/README/ (visited on 02/20/2020).

[30] (Oct. 9, 2014). Manuals-om-20000104, [Online]. Available: https://www.novatel.com/assets/Documents/Manuals/om-20000104.pdf (visited on 02/14/2020).

[31] (Feb. 2, 2020). Serial communication - wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Serial_communication (visited on 02/14/2020).

[32] (Feb. 14, 2020). Gavlab/novatel: Interface for novatel gps receivers, [Online]. Available: https://github.com/GAVLab/novatel (visited on 02/14/2020).

[33] (Feb. 14, 2020). World geodetic system 1984, [Online]. Available: https://www.wikiwand.com/de/World_Geodetic_System_1984 (visited on 02/14/2020).

[34] (Jan. 31, 2020). Oem7_commands_logs_manual, [Online]. Available: https://docs.novatel.com/OEM7/Content/PDFs/OEM7_Commands_Logs_Manual.pdf (visited on 02/17/2020).

[35] (Feb. 2, 2020). Inertial measurement unit - wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Inertial_measurement_unit (visited on 02/17/2020).

[36] (Feb. 9, 2020). Matplotlib: Python plotting — matplotlib 3.1.3 documentation, [Online]. Available: https://matplotlib.org/ (visited on 02/17/2020).

[37] (Feb. 17, 2020). D-link dgs-1008p 8-port gigabit ethernet poe switch — dlinkworks.com, [Online]. Available: http://www.dlinkworks.com/DGS-1008P.asp (visited on 02/17/2020).

[38] (Feb. 17, 2020). Vimba: The sdk for allied vision cameras - allied vision, [Online]. Available: https://www.alliedvision.com/en/products/software.html (visited on 02/17/2020).

[39] (Feb. 10, 2020). Maximum transmission unit - wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Maximum_transmission_unit (visited on 02/17/2020).

[40] (Feb. 17, 2020). Vimba c++ manual cpp, [Online]. Available: https://usermanual.wiki/Document/Vimba20CPP20Manual.1841617038/view (visited on 02/17/2020).

[41] (Feb. 17, 2020). Opencv api reference — opencv 2.4.13.7 documentation, [Online]. Available: https://docs.opencv.org/2.4/modules/refman.html (visited on 02/17/2020).

[42] (Jan. 16, 2020). 045/image - link developers rfc, [Online]. Available: https://draive.com/link_dev/rfc/045/README/ (visited on 03/02/2020).

[43] D. E. King, "Max-margin object detection", *arXiv preprint arXiv:1502.00046*, 2015.

[44] (Dec. 14, 2019). Dlib c++ library - dnn_mmod_face_detection_ex.cpp, [Online]. Available: http://dlib.net/dnn_mmod_face_detection_ex.cpp.html (visited on 02/17/2020).

[45] (Dec. 14, 2019). Dlib c++ library - face_detection_ex.cpp, [Online]. Available: http://dlib.net/face_detection_ex.cpp.html (visited on 02/17/2020).

[46] (Dec. 14, 2019). Dlib c++ library - dnn_introduction2_ex.cpp, [Online]. Available: http://dlib.net/dnn_introduction2_ex.cpp.html (visited on 02/17/2020).

[47] (Dec. 14, 2019). Dlib c++ library - layers_abstract.h, [Online]. Available: http://dlib.net/dlib/dnn/layers_abstract.h.html (visited on 02/17/2020).

[48] (Feb. 10, 2020). Rectifier (neural networks) - wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Rectifier_(neural_networks) (visited on 02/18/2020).

[49] (Feb. 18, 2020). Lighttransport/prnet-infer: C++11 port of yadiraf's prnet(inference only), [Online]. Available: https://github.com/lighttransport/prnet-infer (visited on 02/18/2020).

[50] (Jan. 9, 2020). Importing a tensorflow graphdef based models into tensorflow.js, [Online]. Available: https://www.tensorflow.org/js/tutorials/conversion/import_saved_model (visited on 02/18/2020).

[51] (Feb. 14, 2020). Tensorflow::clientsession class reference — tensorflow core v2.1.0, [Online]. Available: https://www.tensorflow.org/api_docs/cc/class/tensorflow/client-session (visited on 02/18/2020).

[52] (Jan. 16, 2020). Webcam - link developers rfc, [Online]. Available: https://draive.com/link_dev/rfc/017/README/ (visited on 02/20/2020).

[53] (Feb. 20, 2020). Cmake-packages(7) — cmake 3.17.0-rc1 documentation, [Online]. Available: https://cmake.org/cmake/help/latest/manual/cmake-packages.7.html (visited on 02/20/2020).

[54] (Feb. 20, 2020). Find_package — cmake 3.0.2 documentation, [Online]. Available: https://cmake.org/cmake/help/v3.0/command/find_package.html (visited on 02/20/2020).

[55] (Feb. 20, 2020). Exporting and importing targets · wiki · cmake / community · gitlab, [Online]. Available: https://gitlab.kitware.com/cmake/community/-/wikis/doc/tutorials/Exporting-and-Importing-Targets (visited on 02/20/2020).

[56] (Feb. 20, 2020). Set_property — cmake 3.0.2 documentation, [Online]. Available: https://cmake.org/cmake/help/v3.0/command/set_property.html (visited on 02/20/2020).

[57] S. Lowe. (Jan. 1, 2020). Using the fork-and-branch git workflow - scott's weblog - the weblog of an it pro focusing on cloud computing, kubernetes, linux, containers, and networking, [Online]. Available: https://blog.scottlowe.org/2015/01/27/using-fork-branch-git-workflow/ (visited on 02/12/2020).

[58] Atlassian. (Feb. 12, 2020). Forking workflow — atlassian git tutorial, [Online]. Available: https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow (visited on 02/12/2020).