Norwegian University of Science and Technology

Project thesis

TTK4550 Cybernetics and Robotics
Department of Engineering Cybernetics

Pose estimate for quadcopter using stereo camera

Author
Håkon Larsen Eckholdt

Supervisor
Prof. Tor Engebret Onshus,
IE

December 18, 2018

NTNU
Norwegian University of
Science and Technology

Abstract

This project thesis is a continuation of an ongoing Lego-robot project, comprised of several robots that cooperate on solving a maze. Each robot has different sensors and information, and combing the information from all the robots in a Java server application should make the robots able to solve the maze. This project focus on the task of making a drone that maps the labyrinth from above. At this stage in the process, the main task is to implement a system to replace GPS localization, as the maze exists in an indoor environment. This project is a continuation of the work performed by Bendik Iversen(1), which currently consists of a visual odometry algorithm that to some extent manages to track the location of a stereo camera. This project thesis aims to find the best way to proceed in order to achieve better pose estimates than the algorithm currently does. In order to determine this, I have looked at alternative paths to achieve visual odometry pose estimates through a study of different theories and current algorithms in the computer vision universe. I have also updated the instructions on how to install the software and hardware drivers, and changed some libraries in the code in order to make the system usable again, after the OpenCV and Duo team made changes during the summer, rendering the system unusable.

I suggest that the current algorithm is changed to implement a stricter indirect method (2.2.1), using the ORB (2.2.1.3) feature detection and feature matching algorithm to achieve a much better performance than the current algorithm. The system currently struggles with fast movement and the false assumption that the environment is stationary. Taking the decision to change from feature tracking to feature matching (ORB) eliminates this problem, and I conclude that it is a better option for this system. The SVO algorithm could be considered if the ORB change is insufficient, but I find the expected reward to be too low for it to be the primary objective. I also conclude that the focus should be on finding the best accuracy, as opposed to increasing speed to the point where it would have a high enough update frequency to maintain stable flight using only the pose estimates as input to the PID controller. Instead it should replace only the GPS signal, and other common means such as IMU should handle the stable flight of the drone. Lastly I conclude that bundle adjustment should be included, as the system currently relies on raw single frame pose estimates which has proven to be too unreliable.
The findings can be summarized as follows:

- The system should be based on an indirect approach, using features and feature matching as the means of determining pose.
- The feature tracking algorithm should be removed and replaced with feature matching using the ORB feature detector.
- IMU control of the drone should be implemented, and local bundle adjustment should be added to the algorithm.

Preface

The basis for the work done in this project is the master thesis written by Bendik Bjørndal Iversen(1), and the accompanying source-code for a visual odometry algorithm that to some degree manages to track the position of the camera. The source code was made for a TX1 which I also had access to during the project, and a Duo M stereo-camera which account for the camera who's position the algorithm is to track. However, the algorithm and the Duo M camera would not work when I received it due to changes in the libraries and drivers between the time the master was delivered the time I received the code. The theory described in this project is either acquired from sources online cited where used, or from different computer vision related subjects I have completed before and parallel to this project (TDT4265 and TTK21 respectively). The list of hardware I have had access to and use for is listed below, and it was received at the very beginning of the semester.

- Jetson NVIDIA TX1 with accompanying developer board
- Power supply, SD-card and peripheral devices such as keyboard and screen
- Duo M stereo camera
- Nordic BLE dongle
- Desktop for setup.

During this project I have had help from my supervisor in determining the right path of my work, what areas to focus my time and who to contact when special expertise was needed. On account of that I received important help from the department engineer at the department of engineering cybernetics when the Duo M camera drivers were no longer working on the TX1.

Abbreviations

API	=	Application Programming Interface
BLE	=	Bluetooth Low Energy
BRIEF	=	Binary Robust Independent Elementary Features
CPU	=	Central Processing Unit
CV	=	Computer Vision
FAST	=	Features from Accelerated Segment Test
GPS	=	Global Positioning System
GPU	=	Graphics Processing Unit
HOG	=	Histogram of Oriented Gradients
IMU	=	Inertial Measurement Unit
I/O	=	Input / Output
MAV	=	Micro Aerial Vehicle
ORB	=	Oriented FAST Rotated BRIEF
PID controller	=	Proportional-Integral-Derivative controller
RANSAC	=	Random Sample Consensus
rBRIEF	=	rotation-aware BRIEF
SDK	=	Software Development Kit
SIFT	=	Scale Invariant Feature Transform
SoC	=	System on a Chip
SVO	=	Fast Semi-Direct monocular Visual Odometry
USB	=	Universal Serial Bus
VO	=	Visual Odometry

Contents

Abstract **i**

Preface **ii**

Abbreviations **iii**

1 Introduction **1**
1.1 Background and previous work 1
1.1.1 Lego-robot project 1
1.2 Problem description 1

2 Theory **2**
2.1 Visual odometry 2
2.2 Direct vs Indirect 3
2.2.1 Indirect method 4
2.2.2 Direct method 11
2.3 Stereo camera 16
2.4 Local bundle adjustment 18

3 Software Setup **20**
3.1 Run on a Linux system 20
3.2 Installing Drivers for Duo stereo camera 20
3.3 Run from external SD card 20
3.4 OpenCV 21

4 Hardware **22**
4.1 Hardware specification 22
4.1.1 Nvidia Jetson TX1 22
4.1.2 Duo M Stereo Camera 23
4.1.3 nRF51422 BLE dongle 24
4.1.4 SD card 24
4.2 System architecture 25
4.3 Running the system from SD-card 26
4.4 Complete hardware setup 27

5 Discussion **28**
5.1 Direct vs Indirect 28
5.2 Update Frequency 29
5.3 Bundle Adjustment 29

6 Future Work **30**
6.1 Visual Odometry Algorithm 30
6.1.1 Implementing ORB feature detection and matching 30
6.1.2 Local bundle adjustment 30
6.1.3 Tuning and increased speed 30
6.1.4 Outlier removal and distributed features 31
6.2 Complete Drone System 31
6.2.1 Maze mapping algorithm 31
6.2.2 Drone 31
6.2.3 TX1 Carrier board 31
6.2.4 Server communication 31

A Installation Instructions **34**
A.1 Install the right gcc and g++ 34
A.2 Install OpenCV with OpencCV_Contrib 34

B Description of attachments **36**
B.1 Source Code 36
B.2 Previous reports and theses 36
B.3 Datasheets 36

1 Introduction

1.1 Background and previous work

1.1.1 Lego-robot project

The ultimate goal of the work conducted in this project is to develop a drone that knows where it is and how it is oriented in a room containing a maze and several other ground-based robots. The drone should also be able to find and map wall-segments. The information of both the position of the drone and the map of the maze should in turn be sent to a server running a Java-application through the use of BLE (Bluetooth Low Energy). The Server will make a map of the maze using information from not only the drone, but all the other robots in the maze.

1.2 Problem description

Currently the system can find the relative position of the drone to a certain degree. The system fails during fast movement, and even when moved slowly the position estimation is drifting. It's apparent that the frame rate needs to be increased, or the algorithm changed to accommodate the bigger movement between frames. Considering that I want the drone to be able to sustain flight in a contained area for a large amount of time, any drift at all might be catastrophic, and the algorithm should be adjusted to minimize drift or otherwise overcome this problem.
This project aims to find some potential solutions to increase the performance of the current system(1), and determine whether or not the currently implemented solution is the most promising path to a fully functional in-door drone positioning system. The focus of the thesis revolves around the theoretical aspect of the different paths to achieve such an algorithm, and will guide the further development of the lego-project by giving strict guidelines on what areas of VO the future labor should be focused on in order to achieve a positioning system that we can eventually fit on a drone. This project does not intend to further develop the software, however, it will provide useful insight into why different changes and future additions to the algorithm should be made. It is assumed that visual odometry is the way to go in this project, and no attempt will be made to find other options.
The main questions I aim to answer are the following:

- Should the algorithm be based around the direct or the feature-based approach?
- Is feature tracking, as the current algorithm relies on, the best approach, or does another strategy seem more promising?
- Are there additional features that would improve the performance of the tracking?

2 Theory

Odometry is the technique of using data from a sensor to determine the position of an object as a result of the change in position over time(2). Traditionally this is performed using some sort of wheel or rotational encoders dragging or moving across a surface. For instance a wheel might be mounted on the back of a dog-sled in order to know the distance one have traveled in a particular direction. This is, however, prone to errors as the object of interest might slip and otherwise react unpredictable to changes in climate or other external stimuli. In this particular paper the reason why traditional odometry is not ideal is quite obvious: The drone will never be in contact with any surface during operation. This is where visual odometry comes into play.

2.1 Visual odometry

Visual odometry, much like traditional odometry is a means to establish a location by measuring the change in position over time. Unlike regular odometry visual odometry don't rely on tracing the movement of moving parts across a surface, but rather the movement that had to occur to account for the difference between two images taken at two different instances of time. This process is also being used on ground-based vehicles since you remove the uncertainty that arise from drift caused by slipping in regular odometry. As computers get faster and methods in artificial intelligence excel, these methods effectiveness increase. An essential part of visual odometry is the ability to recognize objects or features in multiple different images, so that you can calculate how the object has changed in size, position or orientation from one image to the next. This is done using algorithms from the field of computer vision.
A drone such as a quadcopter has six degrees of freedom (6 DoF). Up/down, right/left, forwards/backwards, yaw, pitch and roll, as you can see in figure 1. The visual odometry algorithm will have to know how the camera moves in all these directions..

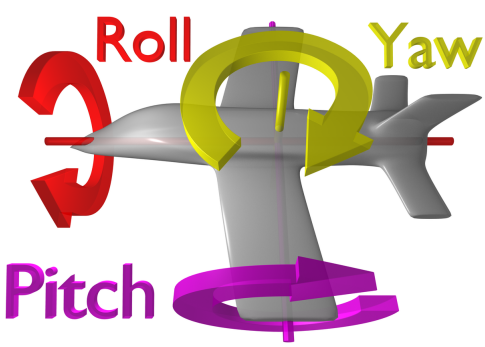


Figure 1: Roll, yaw and pitch of an air-frame (3)

2.2 Direct vs Indirect

When referring to Visual Odometry there are two main methods to achieve the goal of estimating the relative pose of the camera, the translation and rotation between the two images, namely the direct method, and the indirect method. They have a significant difference in the approach of estimating movement, and in applications similar to this project there have been a vast majority of direct algorithms, and that is indeed the case for the previous work done on this very project. The reason why I wish to look into the different methods even though the previous master thesis is entirely focused and developed around the indirect method is the promising results of the Robotics and Perception Group (4). They used a semi-direct method for exactly the same problem as we are trying to solve in this project, not considering the communication and mapping of the maze. I believe it is worth looking into whether the direct or indirect method is better suited for this task.

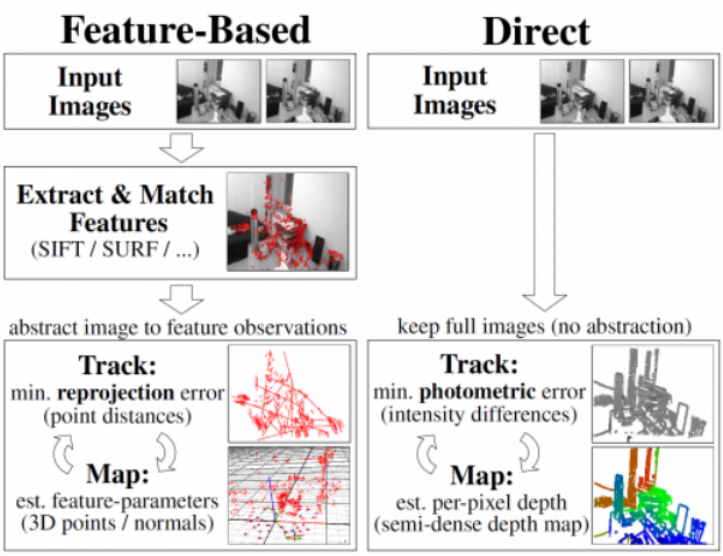


Figure 2: Comparison between Feature-Based and Direct method of estimating position (5)

The main difference is illustrated in figure 2. The main difference is that where you in the Indirect method (appropriately named Feature-Based in the figure) you extract features before estimating the movement of said features, the direct method compares the individual pixels without attention to specific features. The following gives a more comprehensive explanation of the different methods and their strengths and weaknesses.

2.2.1 Indirect method

The indirect method is based around the concept of features. A feature is the combination of a key-point and a descriptor of said point. A key-point is usually a corner or another segment of an image which is easily distinguished from the rest of the image, and would most likely be found again in an image with a slightly different pose. A feature descriptor is some way of describing the point, or rather it's immediate surroundings, making it possible to distinguish different points from one-another, and also match points from one image to another.
The indirect method is all about minimizing the geometric error that occur between the triangulated matching points, and the assumed pose of the camera. When this is optimized over the the pose of the camera the correct pose is approximated. This can be stated as:

$$T_{cw}^* = \underset{T_{cw}}{\operatorname{argmax}} \sum_j ||\pi(T_{cw}\tilde{x}_j^w) - u_j||^2$$

The first element in the sum, $\pi(T_{cw}\tilde{x}_j^w)$, is the prediction part of the equation. This predicts where the points would be on the 2D frame on the image plane given a pose T_{wc}. When minimizing the geometric error of this prediction with the actual value u_j the ideal correct pose is reached. This is the pose of the world as seen by the camera, what we actually want is the opposite:

$$T_{wc}^* = \underset{T_{wc}}{\operatorname{argmax}} \sum_j ||\pi(g(T_{wc}, x_j^w)) - u_j||^2$$

As mentioned in the beginning of this section we are in need of features in order to perform this tracking of the camera movement. I will describe the different parts that goes into this in the following sections.

2.2.1.1 Feature detection A feature, or a key-point, can be classified as a small area of an image that may be found in different images, and the strength of said points depend on how unambiguous they are. If you found the key-point in two images you should have a relatively small distance of potential error. Imagine that you find a white spot on a white wall in two images. You have almost no way of knowing where on the wall the points are, and you have a large amount of uncertainty. This is a bad point. If you find a point on an edge you have a better result, as you know it's along a line, but it could still be anywhere on that line. The best key-points are those who consist of corners. That is not necessarily corners in the traditional sense where two corners clearly meet, but a point in the image where the image intensity has a gradient in all directions. The latter is the key-points we aim to find when using the indirect method. These different points can be seen in figure 3.

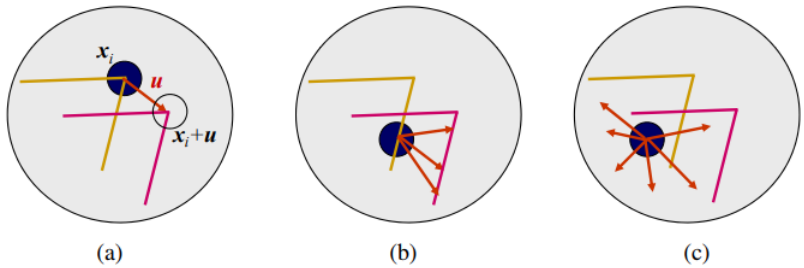


Figure 3: The three main categories of key-points. (a) Corner (b) Edge (c) Flat surface (6) [pp. 210]

There are many different methods of finding key-points, or features. There are multiple considerations when choosing the detector for a project. Speed and accuracy are the most apparent, but there are other qualities such as scale invariance, rotational invariance, affine invariance. Ideally we would achieve a high enough frame-rate so that the difference between two images is small enough for these considerations to be a mere luxury and not necessarily for a good result. This is not necessarily the case, and especially rotation is a lightly distortion as the drone might could be expected to roll to keep its position.
Among the many feature detectors we find the Canny edge detector, Harris corner detection and FAST. I will describe the FAST (Features from accelerated segment test) here.

FAST - Features from accelerated segment test Fast is a corner detector (as opposed to edge or blob detector) which is computationally very efficient. This is the main reason why it is widely used for real time applications, including this project. It is a natural choice as we already see that frame-rate might become an issue with the current algorithm when put to use on a real drone. It is also the feature detector used by ORB(Oriented FAST rotated BRIEF) algorithm, which I will elaborate on further in 2.2.1.3. The FAST algorithm(7) works by addressing a point which is to be classified as a point or not, p. Around this point there is created a Bresenham circle consisting of 16 pixels (see figure 4 for reference). You also define a threshold value t which represents the change in luminosity between the luminosity at the interest pixel p (I_p) and the luminosity at one of the 16 points in the circle needed for the difference to be considered significant. The last thing needed is the number of pixels in the circle who needs to be either sufficiently darker, or lighter than the point p, this number is n. With these values defined the process is as follows:

- Define the point p you want to categorize.
- Check if there consist a set of n points in the circle surrounding p that all fulfill $I_n \leq I_p - t$, or all fulfill $I_n \geq I_p + t$, that is, that all points are either dark enough or light enough.
- If such a set exist p is a corner.

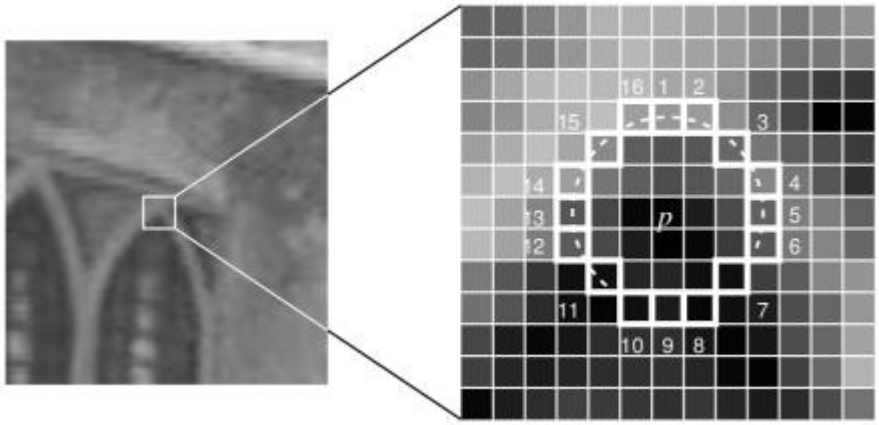


Figure 4: The circle that is evaluated in order to determine if a point is a corner in the FAST algorithm (7)

An attempt at making this even faster is to include a pre-test that first check if the points 1, 5, 9 and 13 has the property of three or more are all brighter or all darker than the luminosity of p plus the threshold. If that is not the case the point can immediately be discarded as not being a corner, and the algorithm saves up to twelve comparisons. However, this poses a few problems:

- If $n < 12$, as you cant discard a corner based on just four comparisons when the set-size is that small.
- The efficiency of the test is dependant on which pixels are compared first.
  - These two first problems can be solved using machine learning.
- Multiple features can be detected adjacent to each other.
  - This can be solved using Non-maximal suppression which essentially is to discard the feature with the least average luminosity difference between the point p and the points in the circle.

2.2.1.2 Feature Matching There are two main ways of determining the correspondence between features from an image to another, feature matching, and feature tracking (2.2.1.4). The main difference between these is that feature matching finds features in two images, and then match them in order to estimate movement by minimizing the geometric error. This method is the cleanest form of the indirect method, and an example is shown in figure 5. Feature tracking on the other hand, only finds features in the first image and then tracks them to the next using a local search technique. This is not necessarily strictly indirect, but we will discuss this later when considering the SVO (Fast Semi-Direct Monocular Visual Odometry) 2.2.2.2 method. As the introduction of the last paragraph dictates the goal of feature matching is

matching features found in image $k-1$ to features found in the current image k. This relies on some sort of method to distinguish the different key-points, other than their relative position, as that by all accounts have changed. The previous key-point algorithm FAST does not provide that, since it simply determines what a corner is. In order to match point we also need a **feature descriptor**. This is something that ideally uniquely identifies a point based on its soundings. If the feature detector does not provide this, we need to add one before matching. There exist multiple, like SIFT, HOG (Histogram of Oriented Gradients) and BRIEF (Binary Robust Independent Elementary Features). The descriptor currently in use in this project is the SIFT descriptor.

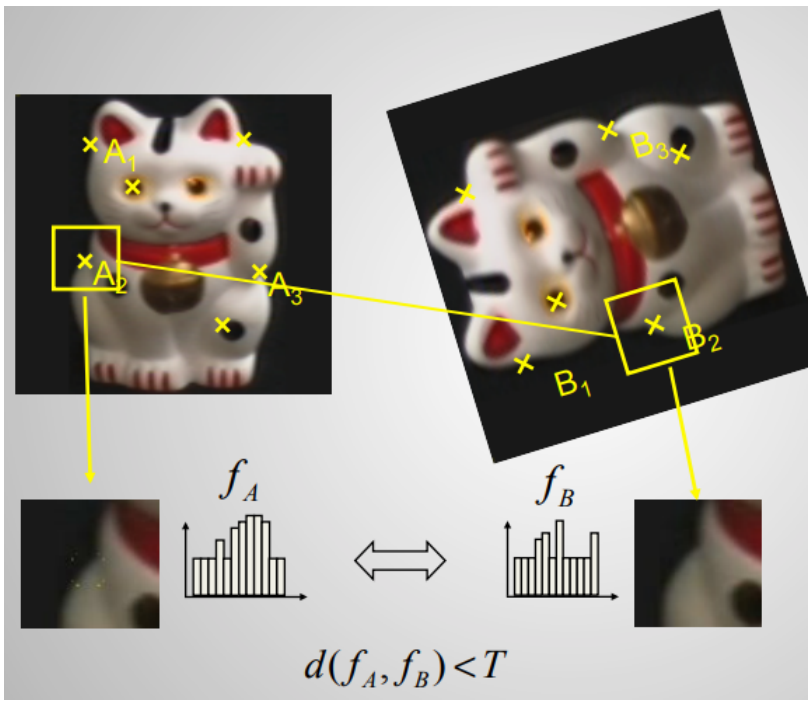


Figure 5: Visualization of feature matching where point A2 and B2 are matched because the difference between the descriptors f_A and f_B are smaller than the threshold value T(8)

SIFT - Scale Invariant Feature Transform The SIFT(9) algorithm creates a normalized 128 dimensional feature vector which can be compared to others, usually using a nearest neighbour approach. The algorithm works as follows:

- Extract a 16x16 patch around a key-point. Divide this into 16 4x4 windows.
- Inside each 4x4 window, calculate the gradients and place them in a histogram of 8 bins.
- Apply a Gaussian weighing function to make the points farthest away from the key-point make less of an impact.

- Collect the histograms of the 16 4x4 windows into a feature vector of 128 (4x4x8) numbers.
- Normalize the vector. The vector looks similar to f_A in figure 5, only with 128 values.

BRIEF - Binary Robust Independent Elementary Features I choose to shortly discuss the BRIEF(10) because it is the descriptor used in the ORB method, which is promising for this project. The BRIEF method is very fast compared to other descriptors as it directly calculates a binary strings from the image patches around the keypoints. Not only is the calculations faster than computing a comprehensive descriptor, but using the Hamming distance (Bit-wise difference between strings) is extremely fast to compute on modern CPU's. The bit string that is the descriptor is comprised of multiple tests within the patch. The test are simply a comparison of the luminosity values of two pixels in the patch. This can be expressed mathematically as

$$\tau(\mathbf{p}; x, y) := \begin{cases} 1, & \text{if } \mathbf{p}(x) < \mathbf{p}(y) \\ 0, & \text{otherwise} \end{cases}$$

with $\mathbf{p}(.)$ representing the pixel intensity at a point and τ is the test result. The numbers of test to compute on a patch is optional, but the original paper(10) has good results with both 128, 256 and 512 tests. (32 and 64 bytes). The binary string is comprised of n_d tests like this

$$f_{n_d}(\mathbf{p}) := \sum_{1 \leq i \leq n_d} 2^{i-1} \tau(\mathbf{p}; x_i, y_i)$$

where as mentioned n_d usually is either128, 256 or 512. The final matching of descriptors is done using

$$L = \sum_{0 \leq i \leq n_d} XOR(f_{n_d}{}^1_i, f_{n_d}{}^2_i)$$

which is the Hamming distance between a patch in image 1 and image 2. If L is smaller than some threshold it's considered a match

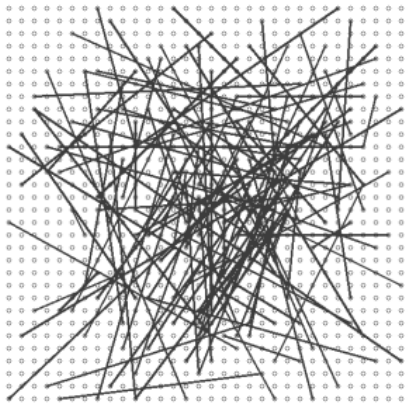


Figure 6: Visualization of tests on in a BRIEF descriptor. Lines stretch between pixels that are to be compared. This particular test distribution is achieved using an isotropic Gaussian distribution.(10)

This was the basics of the BRIEF algorithm, but there are two other aspects to consider, which I will not go further into but should be mentioned.

- The patch should be smoothed out. This is because if this is not done, single pixel-values are used, which are very susceptible to noise.
- The test locations x_i and y_i have many different possibilities, and must obviously remain unchanged between every descriptor. The best results were achieved by using an isotropic Gaussian distribution as seen in figure 6
- The BRIEF feature descriptor does not take orientation into consideration like for instance SIFT.

2.2.1.3 ORB - Oriented FAST Rotated BRIEF ORB(11) is, as the name suggests the combination of finding key-points using the FAST2.2.1.1 algorithm, and then perform feature matching using the BRIEF descriptor. There are some adjustments made to the detection part (FAST) of the algorithm in order to make it more robust. First of all they introduce a Harris corner measure, in order to maximize the number of key-points that represent corners and not edges. This increases the chances of key-points in one image correctly matching points in the next image. Further more the FAST algorithm is not scale invariant. This is taken care of by adding a scale pyramid and producing features on all the different levels. In addition to these modification, it was also a need to implement some sort of rotation awareness. This is done by taking advantage of the intensity centroid. The intensity of centroid is determined by taking the moments of the image patch

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y), p, q \in [0,1]$$

, and from that determine the centroid as

$$C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}})$$

By looking at the moments formula you see that m_{00} simply gives the sum of all non-zero (by intensity) pixels in the patch, or if you like: the area of the patch. The m_{10} and m_{01} is the "mass" of x and y respectively. That implies that C is actually $C = (\overline{x}, \overline{y})$ where $\overline{x}$ and $\overline{y}$ is the average of x and y. The patch orientation is simply determined as $\Theta = 2(m_{01}, m_{10})$.
Following the key-point detection is the matching, and in order to achieve this you need a descriptor. This is where BRIEF comes into play. As mentioned in 2.2.1.2, the BRIEF descriptor will fail if rotation is introduced. ORB solves this by introducing what they call rBRIEF, Rotation-Aware BRIEF. This is the result of using steered BRIEF instead of regular BRIEF, and then running a greedy search on all the possible binary tests. The ones with means close to 0.5, high variance and that also are uncorrelated are chosen by the greedy algorithm. The "steered BRIEF" can be described as the rotated matrix S_Θ, where $S_\Theta = R_\Theta S$, and S is the matrix of all the binary tests in a patch. The rBRIEF allows rotation and thus makes the ORB rotation invariant.

Table 1: Computation of ORB steps

Table showing the computational time on a 640x480 image on an Intel i7 2.8 GHz processor. Results are from the original ORB paper(11).

ORB:	Pyramid	oFAST	rBRIEF
Time (ms)	4.43	8.68	2.12

As seen in table 2.2.1.3 the computational speed on a desktop processor is very fast, and by looking at table 2.2.1.3 ORB proves to be vastly faster to compute than comparable methods. It also outperforms SIFT and SURF in the outdoor tests performed by the team behind ORB when it comes to accuracy.

Table 2: ORB vs SURF vs SIFT

Table showing the computational time on an Intel i7 2.8 GHz processor for the ORB, SURF and SIFT algorithm. It was run on 2686 images at 5 scales, averaging about 1000 features on all the tests. Results are from the original ORB paper(11).

Descriptor	ORB	SURF	SIFT
Time per frame (ms)	15.3	217.3	5228.7

2.2.1.4 Feature Tracking Feature tracking is another way of finding how a feature point has moved between two frames. This method relies on the movement to be small enough so that the points does not move out of the scope of the chosen local search method. It could be stated that slow motion is an option in this project, if the

multirotor is stabilized through other means, which will be discussed later. However, Iversen(1) was under the impression that the movement might still be too large, and that the moving robots breaks some of the assumptions one must make. Never the less, this is the currently implemented solution in the algorithm, and I will present it briefly.
One way to achieve feature tracking is using **optical flow**(12). Optical flow is the method of estimating movement of the surroundings, or perceived movement of the surroundings caused by camera-movement, by tracking the movement of features from one frame in time, to the next frame in time. As I mentioned in the previous paragraph there are some assumptions to this method that are potential trip-wires if used in this project's application.

- The pixel intensity is assumed constant between successive frames.
- Neighbouring pixels have similar motion
- The motion between successive frames are small

Optical works on the principle of the optical flow equation. It is based on the fact that, given the assumptions, you have

$$I(x, y, t) = I(x + dx, y + dy, t + dt)$$

which using Taylor series expansion can be shown to be

$$f_x u + f_y v + f_t = 0$$

given

$$f_x = \frac{\partial f}{\partial x}; f_y = \frac{\partial f}{\partial y}$$

$$u = \frac{dx}{dt}; v = \frac{dy}{dt}$$

This can't be solved without another method, like for instance **Lukas-Kandle**. Lucas-Kanade uses the assumption that neighboring pixels move similarly, by looking at a 3x3 pixel square and finding (f_x, f_y, f_t) for these 9 pixels. Using the Least squares fit method which ultimately yields:

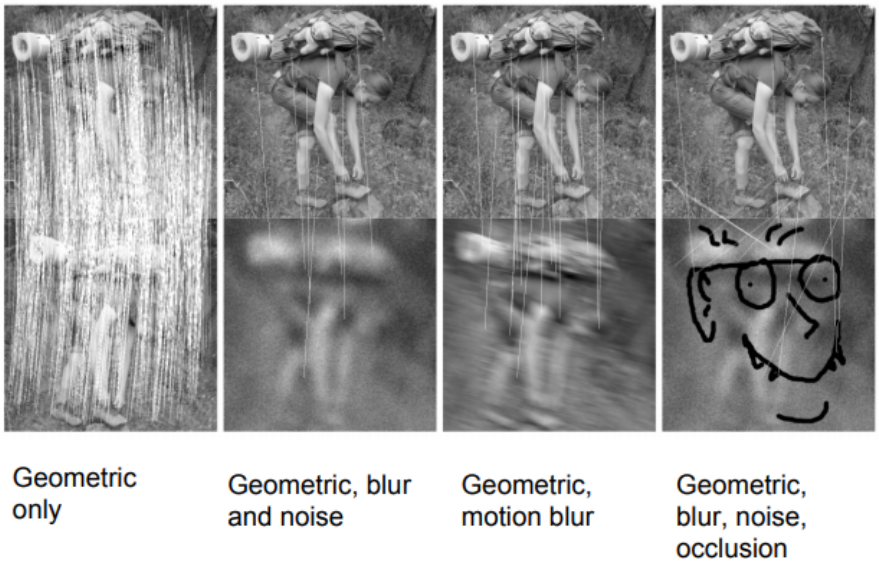
$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i f_{x_i}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i f_{y_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix}$$

Solving this equation of two unknowns above gives the answer to the problem.

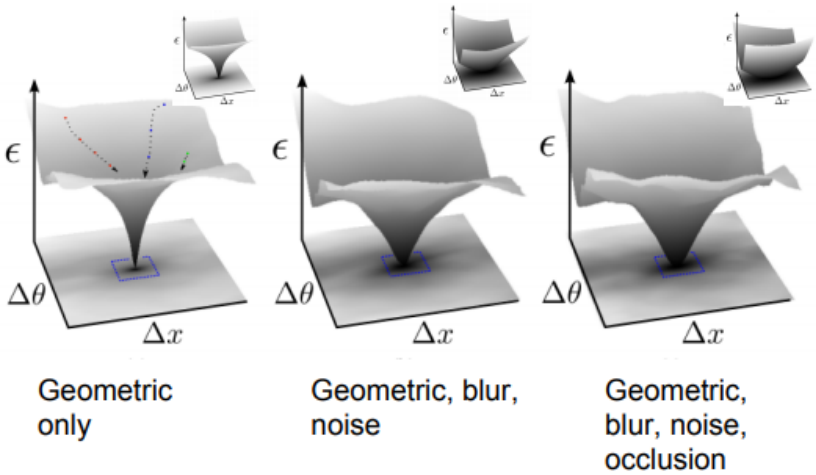
2.2.2 Direct method

In the previous chapter i presented some of the theory behind the indirect method of estimating movement using consecutive frames of a camera. In contrast to the indirect method which is based around the concept of features, the direct method work on individual pixels of the entire image. As seen in figure 2, the direct method skips the step of extracting features and descriptors. The following step of estimating the

pose over all the pixels is, however, more computationally heavy than doing so over a potentially sparse set of features. This method is not seen as often in settings similar to those of this project due to the fact that more computational power is needed, resulting in either bigger computers with a higher power consumption, or slower frame-rate. It might in fact prove impossible to create a real-time VO based on this if the computer is not sufficiently powerful. However, there are some advantages to this method. Advantages described in the next paragraph that might be very useful on a moving airborne platform such as the indoor drone in this project.



(a) White lines represent features matched when different image degradation's are introduced



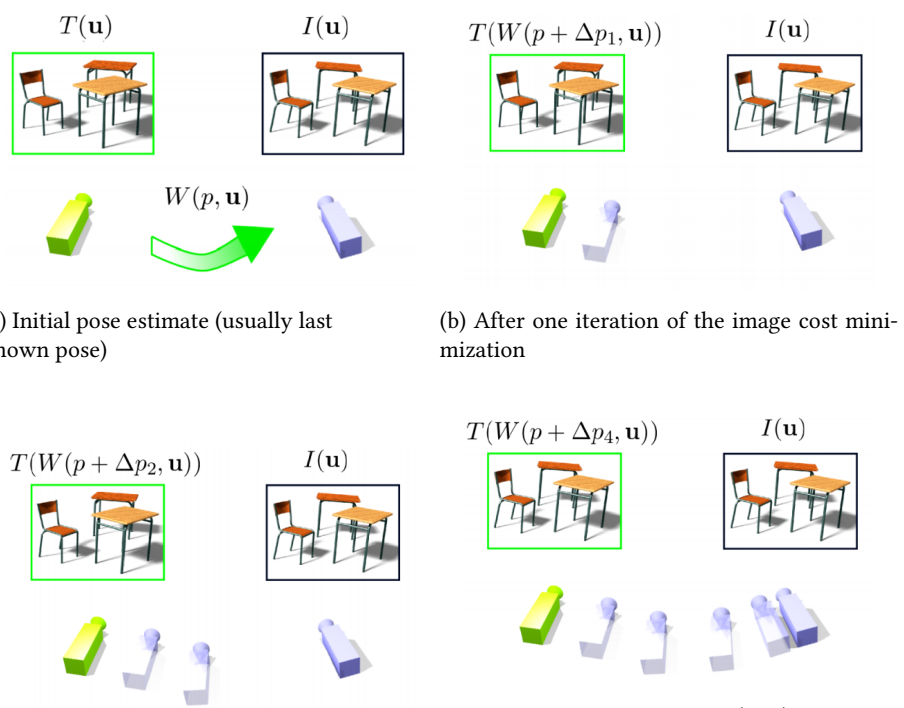
(b) The direct method still manages to find clear global minimum

Figure 7: Illustrations to show that the direct method is much more robust to image degradation such as motion blur, blur and noise. (13)

If there is induced blur, noise, motion blur or other forms of image degradation a

feature detector will have a hard time finding features at all, or find features which are bad and impossible to match in a meaningful way. This problem is absent, or at least greatly less prominent using the direct method which is based on individual pixel intensity. This is shown in figure 7. Especially motion blur can be expected as the drone will more often than not be in some sort of movement, even when attempting to stay completely still.
In the direct method you also have the distinction between dense and semi-dense method, which is the distinction between using every single pixel in the image, or only using pixels patches of high gradient. Nevertheless, the main element of the direct method is that the pose is estimated based on pixel intensity directly, as opposed to features. How this is done is presented briefly below.

2.2.2.1 Direct tracking Direct tracking is all about minimizing the photometric error $(I_{k-1}(w(d, \mathbf{u}, \mathbf{T})) - I_k(\mathbf{u}))^2$, using the warp function $w(d, \mathbf{U}, \mathbf{T}) = \pi(\mathbf{T}\pi^{-1}(d, \mathbf{u}))$. In simple terms this can be visualized as seen in figure 8 (a).



(a) Initial pose estimate (usually last known pose)

(b) After one iteration of the image cost minimization

(c) After two iterations of the image cost minimization

(d) Final pose estimate. The $W(p, \mathbf{u})$ that represents the transform from picture 1 to here represents the pose change

Figure 8: Illustration of minimization of image cost using Warp (5)

We see that we begin with a predicted image, what some prediction algorithm

believes is the state of the image($T(\mathbf{u})$). This is often set as the last image. The image to the right of each figure ($I(\mathbf{u})$) is an illustration of how the environment you are observing right now looks like. You also see an arrow showing the warp ($w(p, \mathbf{u})$) that was needed to go from the image $T(\mathbf{u})$ to $I(\mathbf{u})$. The p in the warp function represents the camera parameters, in this lies the pose of the camera. This is what we would like to determine. As the warp function is increasingly perfected by iterating through different camera parameters p, the predicted image $T(W(p + \Delta p_i, \mathbf{u})$ gets more and more similar to the actual image $I(\mathbf{u})$. The way the iterative change is determined is by the function

$$\Delta p_i = \underset{\Delta p}{\operatorname{argmin}} \sum (I(\mathbf{u}) - T(W(p + \Delta p, \mathbf{u})))^2$$

$$p \longleftarrow p + \Delta p_i$$

which is shown in figure 8(b) with one iteration, (c) with two iterations and (d) with four iterations. It's important to notice that we iterate over the camera parameters p, which is why this eventually gives us the camera pose. That is, given that the model is good enough, which we assume it is. In figure (d) you see that the images are almost identical, and the isometric error would be under a given threshold, and the next frame could be calculated the same way.

2.2.2.2 SVO - Fast Semi-Direct monocular Visual Odometry The SVO algorithm is a visual odometry algorithm made specifically for micro aerial vehicles (MAV), using a direct approach with many smaller pathces of gradients rather than a few large which is more common. It finds features using the FAST algorithm, but instead of creating descriptors it uses the sparse direct motion estimation for finding feature correspondences between the images. As the direct approach dictates, it uses minimization of the photometric error to estimate the pose as seen in figure 9.

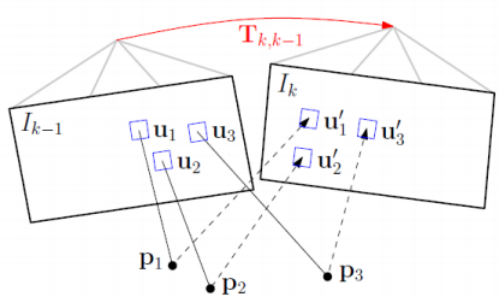


Figure 9: Using sparse image alignment to find the change in pose by minimizing photometric error(4)

The feature matching without descriptors is done by projecting map points from the last to the current frame, and optimizing the position by minimizing the photo-

metric error between the current and the most similar key-frame. This method utilizes motion only bundle adjustment (2.4) in order to estimate the pose of the camera.
The SVO algorithm has a map, consisting of the n last key-frames where n is a fixed number. The map rejects the key-frame that is farthest away from the newest key-frame in distance when a new key-point is introduced. The algorithm takes advantage of depth filters (figure 10), which are initialized for each feature extracted by the FAST algorithm. The depth map is updated by taking the inverse depth, and updating the depth distribution in a Bayesian framework. This is done until the depth variance is low enough, and the algorithm considers it a 3D point.
This differs from the currently implemented Lucas Kandae approach by being computationally much faster because tracking larger distances would require much larger patches than the many small used in this approach, and it would need a pyramidal implantation. You would also need outlier detection. This is not necessary in this algorithm as the spars image alignment ensures no outliers.

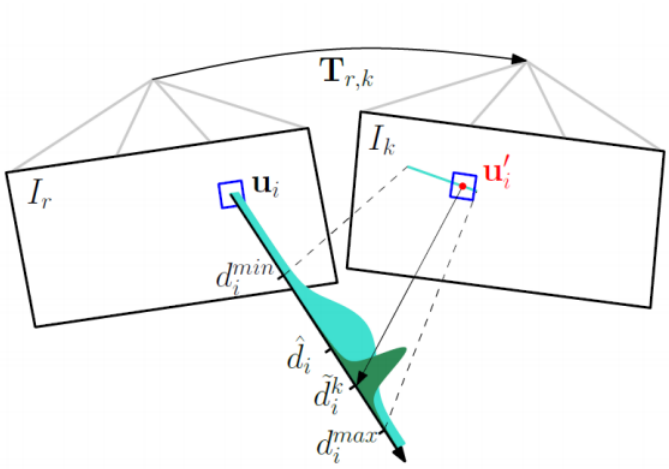


Figure 10: Depth map in SVO algorithm(4)

2.3 Stereo camera

The advantage of using stereo camera as opposed to a single lens-camera is that using a stereo camera gives you the ability do asses the depth of the objects in the image. In figure 11 you see a simple sketch of how the geometry of a stereo-camera work. In this example both cameras point in the same direction, and we assume that the only translation is in the X-direction of the sketch. "b" is the distance between the two lenses, and this is a measurement that has to be known precisely. The depth (in this figure the Z coordinate) can be established using the following formula:

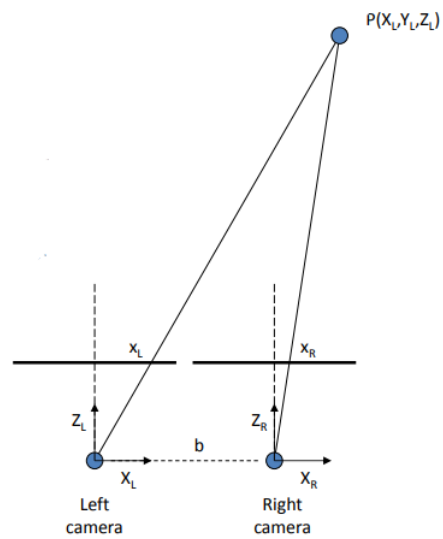


Figure 11: The geometrics of a stereo-camera

$$
\begin{gathered}
x_L = f\frac{X_L}{Z_L}, f\frac{X_R}{Z_R} \\
Z_L = Z_R = Z \\
X_L = X_R + b \\
\implies x_L = f\frac{X_R + b}{Z} \\
where, "d" = Disparity = x_L - x_R \\
d = x_L - x_R = f\frac{(X_R + b) - X_R}{Z} = f\frac{b}{Z} \\
Z = f\frac{b}{d}
\end{gathered}
$$

This is as seen by the math quite a simplification, as we only assume translation in one direction. However, the same principles apply to systems of multiple dimensions. As seen by figure 12 you can find the absolute position by triangulating the position (intersecting the rays). All you need is the intrinsic parameters of the camera (either supplied by the stereo-camera manufacturer, or established using carefull measurements and calibration), the relative pose between the cameras. You then need to find points in the image from the left camera that you can match to points from the right image.

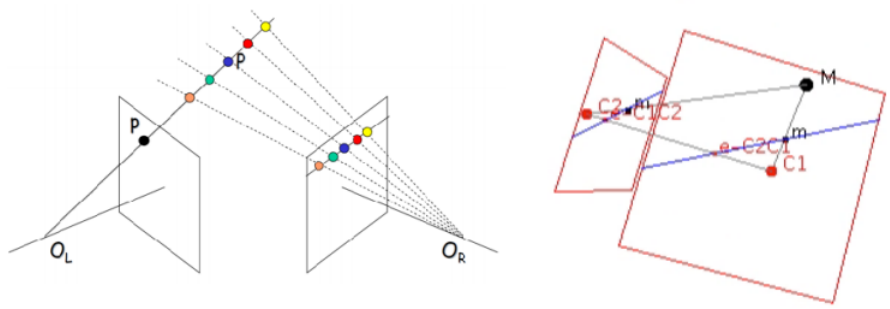


Figure 12: The geometrics of a stereo-camera

2.4 Local bundle adjustment

Bundle adjustment(14) is recommended as part of VO algorithms. It is the concept of estimating the imaging geometry by minimizing the reprojection error with respect to some amount of camera poses and points observed by those cameras. There are different kinds of bundle adjustment; Motion-only bundle adjustment, structure-only bundle adjustment and full bundle adjustment. You also have the option of performing bundle adjustment on the entire set of images after initialization, or to perform local bundle adjustment, that is to perform it on the m last frames. Bundle adjustment can be performed using Gauss-Newton optimization which, for motion only (assuming the camera moves) simplified can be expressed like this: (the process for full and structure bundle adjustment is similar and can be found in (5), as well as the complete calculations on motion-only bundle adjustment.
We wish to minimize the error over state variable $\Theta = \mathbf{T}_{wc}$

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}} \sum_j ||\pi(g(\mathbf{T}_{wc}, \mathbf{x}_j^w)) - \mathbf{x}_{n,j}||^2$$

We then want to, using Taylor expansion, linearize the measurement prediction function

$$h(\mathbf{T}_{wc} exp(\xi_\Delta^\wedge; \mathbf{x}^w) \approx h(\mathbf{T}_{wc}; \mathbf{x}^w) + \mathbf{F}\xi_\Delta$$

, having ξ_Δ being a small perturbation in the camera frame. It can then be shown that we have a linear least-square problem of the form

$$\underset{\xi\Delta}{\operatorname{argmin}} \sum_j ||\mathbf{A}_j \xi_\Delta - \mathbf{b}_j||^2$$

by linearizing around Θ^{it}. This is solved using the normal equations $(\mathbf{A}^T\mathbf{A})\xi_\Delta^* = \mathbf{A}^T\mathbf{b}$ which is using the Gauss-Newton optimization. Then finally update the non-linear estimate Θ^{it+1}

$$\mathbf{T}_{wc} \longleftarrow \mathbf{T}_{wc} exp(\xi_\Delta^{*\wedge})$$

If it has not converged, linearize around the new estimate and repeat. For the complete calculations refer to the slides in (5).

3 Software Setup

3.1 Run on a Linux system

Since I have had difficulties installing the drivers for the duo M stereo camera I decided that I would install it all on a Linux partition on my laptop in order to explore the algorithm there, and revisit the TX1 at a later time. I will not go through how to dual-boot a laptop through different OS partitions, as this is simply the same as running a standard Ubuntu computer. A guide is given on Ubuntu's site(15). It is worth noting that I installed Ubuntu 16.04 LTS as this is the newest supported Linux Version by Duo at this time(16).

3.2 Installing Drivers for Duo stereo camera

The standard procedure for installing the drivers on the TX1 does not currently work, and I had to do a work-around. This is how I installed the drivers: First, download the drivers for the ARM SDK from the duo homepage(17). Run a terminal window on the TX1 (alt + ctrl + t). Type the following into the terminal in order to prepare driver-compilation(18):

```
$ cd /usr/src/linux-headers-'uname -r'
$ sudo make modules_prepare
```

Then open a second terminal and run the code (assuming that the filename for the SDK package is CL-DUO3D-ARM-1.1.0.30):

```
$ cd CL-DUO3D-ARM-1.1.0.30/DUODriver
$ while :;do chmod +x duodriver/run.sh;done
```

Then, in another terminal, compile the driver modules.

```
$ cd CL-DUO3D-ARM-1.1.0.30/DUODriver
$ ./duodriver.run
```

then load the dou-512-ko module. The duo-1024-ko module gives better performance, but it was not compatible with the current setup.

3.3 Run from external SD card

Since the internal memory of the Jetson TX1 is limited to only 16 GB, it is necesary to expand this storage by inserting an SD card into the SD slot on the development board, copying the content of the internal memory and instructing the TX1 to boot from the SD card. The process of doing so is described in a detailed video from JetsonHacks in this article(19). I had to do this again since i re-installed Ubuntu at the beginning of the semester, during debugging of the Duo-driver issues.

3.4 OpenCV

Installing OpenCV is not as straight forward as one might expect. There exist a script for the Jetson TX1 that takes care of the entire installation process for you, and this works straight out of the box(20). This however gives compilation errors when attempting to run the entire system. The reason for this is that the drone-mapping algorithm needs the added library "opencv_contrib"(21). In order to install the OpenCV_contrib library you need gcc-7 and g++-7. This is not standard on ubuntu 16.04.LTS. To install it see Appendix.A.1 You can choose to manually install the openCV and openCV_contrib libraries simultaneously (for this, see appendix A.2) or you can change the "buildOpenCV.sh"(20) file. Change the following lines:

```
16   DOWNLOAD_OPENCV_EXTRAS=NO
to
16   DOWNLOAD_OPENCV_EXTRAS=YES

124  git clone https://github.com/opencv/opencv_extra.git
to
124  git clone https://github.com/opencv/opencv_contrib.git
```

And finally add this line between cmake and -D on line 241. **Change the .. in the line below with the EXACT path**. To find the exact path, type pwd into a terminal at the location where your BuildOpenCVTX1 folder lies.

```
–D OPENCV_TEST_DATA_PATH=../opencv_extra/testdata
```

4 Hardware

This section describes in greater detail than the preface what hardware I have had access to, and that the system relies on in order to function properly. The system architecture is also specified, as well as some considerations that might affect the speed of the system.

4.1 Hardware specification

4.1.1 Nvidia Jetson TX1

The Nvidia Jetson TX1 (or Tegra x1, where Tegra is Nvidia's mobile chip line), is one of the most powerful small full-featured computers on a chip on the market. It is specifically designed to be the go-to choice when performing computer-vision tasks on mobile platforms where size and weight is a concern. The Jetson TX1 should be powerful enough to run a visual odometry algorithm without trouble. The system runs Linux, and comes with a development-board with several I/O connections and features specified in figure 13. The is of the Maxwell architecture with 256 CUDA cores, and 64-bit CPUs. Nvidia has released a new version, Nvidia TX2(22), but the changes do not seem important enough for an upgrade to be considered.

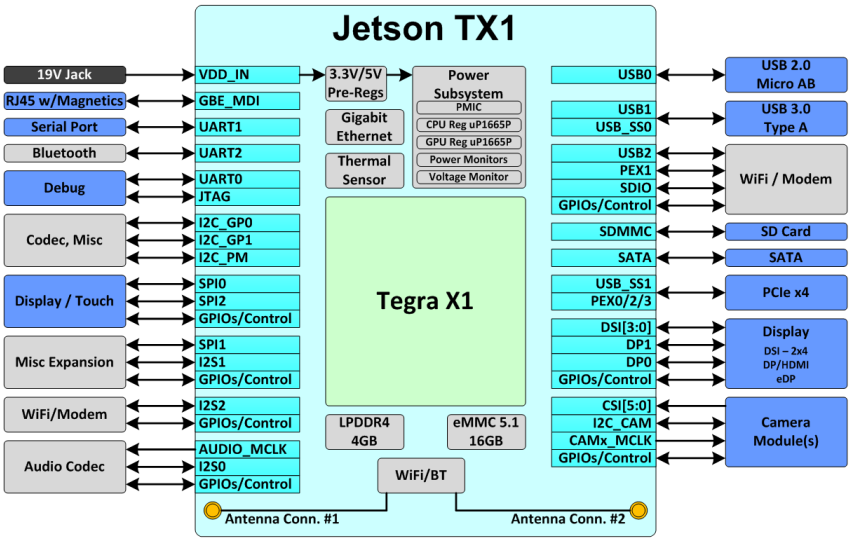


Figure 13: Complete overview of the I/O on the jetson TX1 Dev-board.

The Nvidia Jetson TX1 specifications are listed below, fetched from the Nvidia home-page (23):

- **GPU** | NVIDIA Maxwell, 256 CUDA cores
- **CPU** | Quad ARM A57/2 MB L2

- **Memory** | 4GB 64 bit LPDDR4 25.6 GB/s
- **Data storage** | 16 GB eMMC, SDIO, SATA
- **USB** | USB 3.0 + USB 2.0

This is just a selection of the specifications that i consider important. For the entire specification portfolio visit the Nvidia homepage(23).

4.1.2 Duo M Stereo Camera

The Duo M stereo camera is a small form-factor stereo camera developed by the DUO3D team. Its small size and low weight, combined with a USB interface and high speed makes it well suited for our application. The recommended viewing-distance is also suitable for indoor operation. It also features a Duo SDK (Software Development Kit) and a useful API which makes it easy to integrate into the VO algorithm. Figure 14 shows it's small formfactor, and below you see a list of important specifications from the Duo3D home page.(24)

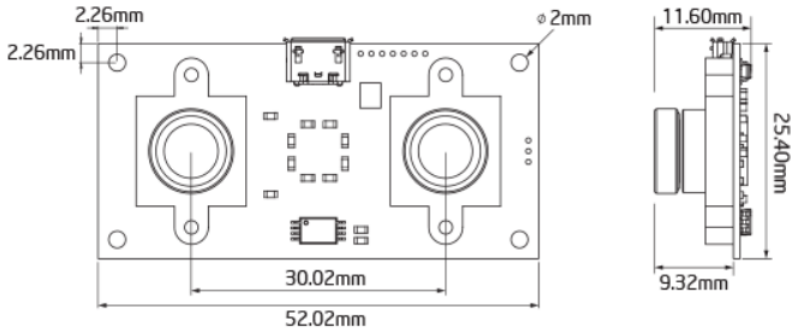


Figure 14: DuoM blueprint with size specifications (24)

- **Dimensions** | 52.02mm x 25.40mm x 11.60mm
- **Weight** | 6.5g
- **Stereo Resolutions** |
  - 45 FPS @ 752x480
  - 49 FPS @ 640x480
  - 98 FPS @ 640x240
  - 192 FPS @ 640x120
  - 86 FPS @ 320x480
  - 168 FPS @ 320x240
  - 320 FPS @ 320x120
- **Pixel size** | 6.0x6.0μm

- **Shutter speed** | 0.3 μsec - 10 sec
- **Baseline** | 30.02mm
- **Recommended depth range** | 0.23-2.5m
- **Field of View** | 165deg Wide Angle Lens with low distortion < 8.5%
- **Focal Length** | 2.0mm - 2.1mm
- **Power consumption** | 2.5 Watt @ +5V DC from USB

4.1.3 nRF51422 BLE dongle

This is the bluetooth dongle intended to communicate with the server. It is developed by Nordic Semiconductor and uses the ultra-low powered SoC nRF51422. The dongle is illustrated in figure 15.

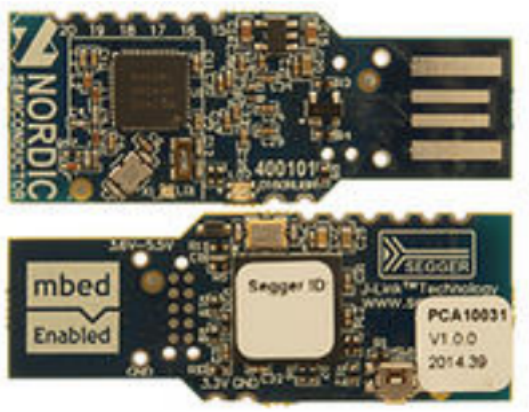


Figure 15: Bluetooth dongle used to communicate with the Java server.

4.1.4 SD card

The internal memory of the TX1 is, as specified in4.1.1, only 16 GB. In order to remove the memory constraint the system is running from a 64 GB micro SD memory card, and accompanying adapter from micro SD to SD. The card is a Class 10 micro SDXC card from Kingston 16. Instructions on how to use external storage as main storage is described in 3.3

Figure 16: The SD-card used when testing the existing code

4.2 System architecture

The system architecture is illustrated in figure 17. There are only a small fraction of the I/O ports that are used for this project, and this indicates that a smaller developer board will suffice for the current set-up. That is a good thing considering that the developer board in this figure, which is the one I currently have access to, is quite large and heavy. Not very well suited for aerial use.

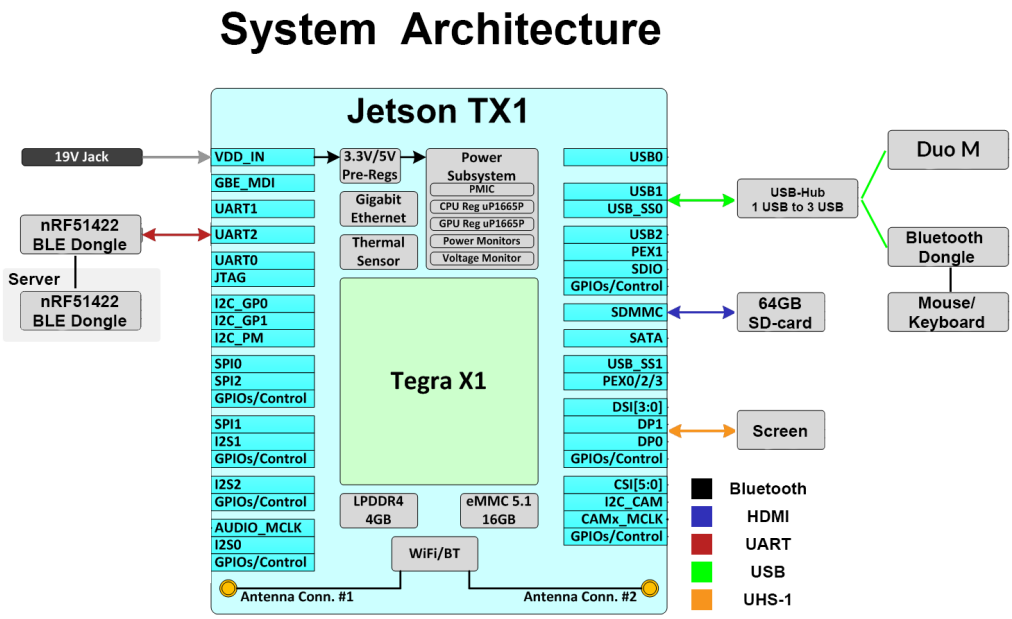


Figure 17: Simplified hardware setup using developer board schematic

4.3 Running the system from SD-card

Figure 18: The SD card on the left was used previously, the one on the right is used this semester

At the beginning of this project i removed the SD-card that was already in the TX1 because i wanted to keep it as backup when i re-installed the entire system. I installed Ubuntu 16.04 using jetpack 3.0 on a clean class 10 SD card formatted as ext4 (Linux format). After using the system for a while I'm sure the system is far slower than what could be expected from the hardware in the TX1. In my opinion the two most apparent reasons for this is either that the SD-card I swapped to is not as fast as the one used earlier. They are both classified as a class 10 card, but that only guarantees 10MB/s sequential write(25). The other might be that the Ubuntu version is too comprehensive (it contains a lot of programs that are never used). It might be worth considering installing everything back on the old SD-card at a later stage. However, the test was sufficient to determine that the algorithm was still working.

4.4 Complete hardware setup

Figure 19: The complete hardware setup

The complete system consist of the Jetson TX1 on the development-board. Mounted to the PCB of the dev-board is a plastic adapter that contains both the USB-hub needed to connect more than one USB-attachment and the DUO stereo-camera. Both the DUO camera and the dongle for wireless mouse and keyboard is attached to the dongle. As seen in the picture19 this allows for a pretty compact solution that is easy to move in the event of testing. Attached is also a Bluetooth dongle. This used to be connected directly to the header-pins of the dev-board because all the USB-ports were occupied. However, this is no longer necessary since the combined keyboard/mouse dongle frees up a port. These connections can be seen in figure 17, where the different kinds of connections are color coded.

5 Discussion

The goal of this thesis has been to determine in what direction the algorithm should be taken, regarding the functions and algorithms it is comprised of. The underlying assumption has always been that visual odometry is the path to success. However, I have explored the theoretical aspect of the algorithms the positioning system is made of today, including others that might be superior. I will attempt to make an informed decision on how to proceed with this project based on the theory and other information presented in this thesis.

5.1 Direct vs Indirect

The biggest conflict in this paper is the choice between an algorithm utilizing the direct or the indirect method. The reason why I was interested in exploring the direct method originates in the success the SVO (2.2.2.2) algorithm has had for a very similar use as the one in this project. However, I do not believe a strict direct method is the way to go. The advantages of resilience against motion blur, occlusion and other image degradation's are certainly very appealing, but I'm afraid the computational expense of a strict direct approach is too great. However, a semi-direct approach such as the SVO can not be ruled out quite so easily. Adapting the semi-direct SVO approach will be a much more drastic step, compared to other solutions, so it might not be worth taking this path before exhausting other options.
The method currently implemented finds features and tracks them using Lucas Kandle, this is in fact fairly close to what the SVO algorithm, but neglecting some of the steps, and it is not working so well. Iversen(1) stated that using feature matching instead of tracking might be a better option. Leaving feature tracking and implementing feature matching would be a step towards a strict indirect method. I believe that could be a good option for the algorithm. Specifically, I find the ORB (2.2.1.3) algorithm to be very promising. Its speed is vastly superior to that of SIFT (2.2.1.2) since it uses the BRIEF(2.2.1.2) algorithm, while introducing rotational invariance. Since the Indirect approach, using features and matching them is the ad-hoc approach to similar problems, I believe discarding it this early would be a mistake. It would also be far easier to implement this in the current algorithm, making it worth testing.
One concern related to positioning systems is that the movement induced from the flying platform may cause some problems. I have mentioned image degradation, but large movements between crisp images would also cause potential problems. Especially when using the direct method, a big pose change between images would be problematic. The indirect method is by far the preferred method if large pose changes are expected between frames, and that is something that can be expected from a drone depending on what speed it moves at. An obvious solution is to simply restrict the drone movement to be slow enough so that the Direct method keeps up, but inflicting restrictions on the moving platform might not be the ideal way of solving this problem. Instead, going for a strict feature-matching based indirect method might be the preferred route.

5.2 Update Frequency

Another consideration is to what extent we should strive towards a high frame-rate. Obviously a fast algorithm that can process the images and estimate the pose in real-time is the goal. However, real time can be defined quite widely, and in computer vision 24 frames pr. second is widely accepted as real time. That would not be enough to keep a drone stable. When controlling a quadcopter you want an update frequency of about 50 Hz simply to keep it marginally stable, and that is obviously not good enough for our purpose, and would be catastrophic even with slight disturbance from ground effect or self-inflicted wind currents in the room.
Another problem is that the VO algorithm was used as the only input to the drone's control system. Any drift at all would make the drone move to an unstable position. One possible solution to this is to introduce the visual odometry algorithm as an input to an autopilot that receives position as a means to determine where to move next, but not in order to maintain safe hover and/or movement. For this the usual update frequency can be as low as 1-10 Hz. This could for instance be done using an on-board gyroscope, or IMU (Inertial measurement unit). This would ensure a stable platform regardless of frame-rate (to a certain extent). It would also secure flight if the drone were to drop frames, not find key-point or if unexpected external forces were to include large amount of image degradation.

5.3 Bundle Adjustment

Bundle adjustment is an important step that could increase the accuracy of the system dramatically, at the cost of heavier computations. It is a trade off that most systems used other places at this time accepts, and will most lightly be even more widespread at computational power increase. Implementing it in this system is currently not tested, so how severely it will restrict the update frequency and how much better the pose estimates will be is unknown, but it is probably worth implementing.

6 Future Work

6.1 Visual Odometry Algorithm

The next step in this project would be to improve upon the visual odometry algorithm, and to implement the changes and improvements to the current algorithm that I have found to be most promising through this theoretical study. The algorithm must be improved to be both more accurate and faster. Until the VO algorithm is reliable attaching it to a flying platform is out of the question.

6.1.1 Implementing ORB feature detection and matching

As i have discussed in this paper, i believe the ORB 2.2.1.3 feature detection and matching is the most promising adaptation of the current algorithm. Considering that the current algorithm fails when moving fast on a controlled, 2 DOG platform (1), I believe tuning and improving the current platform might be more cumbersome than implementing an algorithm that is more robust to large movement and dynamic images. Even if it turns out to be a worse solution in practice, it is worth attempting in order to compare it with the current implementation. In other words, the feature tracking step of the algorithm should be swapped for a feature matching step, specifically that of orb. And the FAST feature detector should include the rotational awareness as described in the ORB section.

6.1.2 Local bundle adjustment

Local bundle adjustment is, as mentioned in section 2.4, an important step to ensure precision in visual odometry algorithms, and after implementing the indirect method ORB, it is definitely something that should be a priority. Tests should be performed to see how many previous images should be included to maximize precision vs computation time. It will most lightly improve the algorithm significantly.

6.1.3 Tuning and increased speed

The algorithm as of now runs at about 5-10Hz (1), which as discussed in 5.2, is enough to replace GPS signals, but is insufficient for controlling the drone as a single input to the controller. Therefor speed either need to be improved, or the more appealing and reliable option: Add some other input to the controller, i.e. IMU. either way, optimization of the code should be kept in mind during all future development, and some of the functions in the current algorithm uses the basic version, while the must faster CUDA option is available. When changing out the algorithms and implementing new, CUDA options should always be used when possible.
Tuning of the parameters should also be done in order to increase performance, but that is not the immediate priority, as the main functionality most lightly has a vastly better performance increase. Fine tuning should be done after the algorithm has been implemented using the best available CV methods.

6.1.4 Outlier removal and distributed features

As Iversen described in his master thesis(1), he found that the image features should be spread more evenly across the image captured by the camera. The effect of implementing this should be tested once the feature matching algorithm is implemented. Further more, he believed that RANSAC or some other outlier detection should be used as opposed to the current inlier detection. The effects of this should also be tested.

6.2 Complete Drone System

6.2.1 Maze mapping algorithm

An essential part of the complete drone system is that the drone manages to map the wall segments of the maze it flies above. This has not been the focus for me, nor has it been the focus of the master thesis I have based my work on(1). An algorithm was developed back when the system ran on a Raspberry PI, but that code is incompatible with the current hardware. In the future this has to be addressed so that the drone not only flies, but can collect useful information for the other Lego robots.

6.2.2 Drone

At one point, a drone has to be acquired to test that the mapping and positioning system works in a real test environment. The drone should be able to carry all the necessary hardware (TX1, cameras, batteries ++) while at the same time being small enough to fly indoors. Obviously it has to be programmable so that it can be flown using our algorithm. The specifications needed are easier to determine when the algorithms performance and final hardware is determined at a later stage in the process.

6.2.3 TX1 Carrier board

As i mentioned in 4.2, the current development board is bulky and heavy, as well as much more comprehensive than we need. Before mounting the TX1 to a drone, the dev-board should be swapped out for a smaller carrier board with only the essential features. What those features are must be determined at a later stage. A smaller board minimize change of destruction in a crash, and makes it possible for the drone to be smaller and cheaper.

6.2.4 Server communication

The server connection is somewhat stable at the current time, and manages to send pose estimates to the server. In the future the drone must also be able to receive commands from the server, such as where to map next. Eventuality some effort should be made towards making the server connection stronger. The wall segments found in the future mapping system should also be sent to the server. This must also be implemented in the future.

References

[1] B. B. Iversen, "Stereo visual odometry for indoor positioning." [Online]. Available: https://brage.bibsys.no/xmlui/handle/11250/2560149

[2] "Odometry." [Online]. Available: https://en.wikipedia.org/wiki/Odometry

[3] "Roll, yaw and pitch of an air-frame." [Online]. Available: https://en.wikipedia.org/wiki/Degrees_of_freedom_(mechanics)#/media/File:Flight_dynamics_with_text.png

[4] "Svo: Fast semi-direct monocular visual odometry." [Online]. Available: https://www.ifi.uzh.ch/dam/jcr:e9b12a61-5dc8-48d2-a5f6-bd8ab49d1986/ICRA14_Forster.pdf

[5] T. V. Haavardsholm, "Short term tracking, pp in ttk21 2018 ntnu." [Online]. Available: https://bit.ly/2BTQBnN

[6] R. Szeliski, *Computer Vision: Algorithms and Applications*, 2010. [Online]. Available: http://szeliski.org/Book/

[7] "Fast algorithm for corner detection." [Online]. Available: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_fast/py_fast.html

[8] T. V. Haavardsholm, "Feature detection, description and matching, pp in tdt4265 2018 ntnu." [Online]. Available: https://bit.ly/2BTQBnN

[9] "Distinctive image features from scale-invariant keypoints." [Online]. Available: https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

[10] "Brief: Binary robust independent elementary features*." [Online]. Available: https://www.cs.ubc.ca/~lowe/525/papers/calonder_eccv10.pdf

[11] "Orb: an efficient alternative to sift or surf." [Online]. Available: http://www.willowgarage.com/sites/default/files/orb_final.pdf

[12] "Optical flow opencv." [Online]. Available: https://docs.opencv.org/3.3.1/d7/d8b/tutorial_py_lucas_kanade.html

[13] U. o. W. Richard Newcombe, "Introduction to dense visual camera tracking." [Online]. Available: http://frc.ri.cmu.edu/~kaess/vslam_cvpr14/media/VSLAM-Tutorial-CVPR14-P12-DenseVO.pdf

[14] "Bundle adjustment." [Online]. Available: https://en.wikipedia.org/wiki/Bundle_adjustment

[15] U. team, "Create a bootable usb stick on ubuntu." [Online]. Available: https://tutorials.ubuntu.com/tutorial/tutorial-create-a-usb-stick-on-ubuntu#0

[16] "Ubuntu releases." [Online]. Available: http://releases.ubuntu.com/?_ga=2.65683559.816906049.1540734196-1742439413.1540734196&fbclid=IwAR063ynVeevodDnhjOUh9lqW-ZqxeScFKETQAqYKMj7GWHwM32uJLrHO8is

[17] "Installation instructions duo sdk." [Online]. Available: https://duo3d.com/docs/articles/install-all

[18] "How to prepare drivers for duo-install." [Online]. Available: https://base.xsens.com/hc/en-us/community/posts/211806629-MTi-300-not-working-on-NVIDIA-TX1-platform-Kernel-3-10-96-

[19] "Jetson from sd card." [Online]. Available: https://www.jetsonhacks.com/2017/01/26/run-jetson-tx1-sd-card/

[20] "Github repository for opencv installation on jetson tx1." [Online]. Available: https://github.com/jetsonhacks/buildOpenCVTX1.git

[21] "Github opencv contrib library." [Online]. Available: https://github.com/opencv/opencv_contrib.git

[22] "Nvidiatx2." [Online]. Available: https://developer.nvidia.com/embedded/buy/jetson-tx2

[23] "Nvidiatx1 spec." [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/

[24] "Duom spec." [Online]. Available: https://duo3d.com/product/duo-mini-lv1#tab=specs

[25] "Speed classification sd cards." [Online]. Available: https://www.sdcard.org/developers/overview/speed_class/

[26] "How to install gcc-7 on ubuntu 16.04." [Online]. Available: https://gist.github.com/jlblancoc/99521194aba975286c80f93e47966dc5

A Installation Instructions

A.1 Install the right gcc and g++

In order to install the openCV_contrib module you need a newer gcc and g++ than the gcc/g++ 5 that is standard with ubuntu 16.04 LTS. I found that version 7 is stable and works on the contrib module. (you need to be able to compile cxx 11, which is why gcc-5 is not adequate.) to install gcc-7 and g++-7 do the following(26):

```
$   sudo add-apt-repository ppa:ubuntu-toolchain-r/test
$   sudo apt update
$   sudo apt install g++-7 -y
$   sudo apt install gcc-7 -y
$   sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-7 60 --slav
```

and then finally, change to the newly installed gcc and g++ by typing

```
$   gcc sudo update-alternatives --config gcc
```

press the number representing the gcc 7 and then press enter.

A.2 Install OpenCV with OpencCV_Contrib

Make sure you have installed gcc-7. open a terminal from your desired location, i.e. home. Update your ubuntu packages:

```
$   sudo apt-get update
$   sudo apt-get upgrade
```

Then, install all the required packages:

```
$   sudo apt-get install build-essential cmake pkg-config git
$   sudo apt-get install libjpeg8-dev libjasper-dev libpng12-dev
$   sudo apt-get install libtiff5-dev
$   sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libdc1394
$   sudo apt-get install libxine2-dev libv4l-dev
$   sudo apt-get install libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev
$   sudo apt-get install libqt4-dev libgtk2.0-dev libtbb-dev
$   sudo apt-get install libatlas-base-dev
$   sudo apt-get install libfaac-dev libmp3lame-dev libtheora-dev
$   sudo apt-get install libvorbis-dev libxvidcore-dev
$   sudo apt-get install libopencore-amrnb-dev libopencore-amrwb-dev
$   sudo apt-get install x264 v4l-utils
```

Download the two repositories openCV and openCV_contrib using the following commands:

```
$   sudo git clone https://github.com/opencv/opencv.git
$   cd opencv
$   git checkout 3.3.0
$   cd ..
```

```
$   sudo git clone https://github.com/opencv/opencv_contrib.git
$   cd opencv_contrib
$   git checkout 3.3.0
$   cd ..
```

Then, compile using CMAKE. The most important part here is to make sure you have installed and activated the right version of gcc (gcc −version) and that the flag "OPEN_CV_EXTRA_MODULES_PATH" is the first of the flags. The path also needs to be complete, .. and should not be a part of the path. The path should be the path to your opencv_contrib/moules loaction. The location can be found by typing pwd in terminal.

```
cmake
    -D OPENCV_EXTRA_MODULES_PATH=/home/nvidia/opencv_contrib/modules \
    -D CMAKE_BUILD_TYPE=Release \ -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D BUILD_PNG=OFF \ -D BUILD_TIFF=OFF \ -D BUILD_TBB=ON \
    -D BUILD_JPEG=OFF \ -D BUILD_JASPER=OFF \ -D BUILD_ZLIB=OFF \
    -D BUILD_EXAMPLES=ON \ -D BUILD_opencv_java=OFF \
    -D BUILD_opencv_python2=ON \ -D BUILD_opencv_python3=OFF \
    -D ENABLE_PRECOMPILED_HEADERS=OFF \  -D WITH_V4L=ON \ -D WITH_QT=ON \
    -D WITH_OPENGL=ON \ -D WITH_OPENCL=OFF \ -D WITH_OPENMP=OFF \
    -D WITH_FFMPEG=ON \ -D WITH_GSTREAMER=OFF \ -D WITH_GSTREAMER_0_10=ON \
    -D WITH_CUDA=ON \ -D WITH_GTK=ON \ -D WITH_VTK=OFF \ -D WITH_TBB=ON \
    -D WITH_1394=OFF \ -D WITH_OPENEXR=OFF \
    -D CUDA_TOOLKIT_ROOT_DIR=/usr/local/cuda-8.0 \ -D CUDA_ARCH_BIN=5.3 \
    -D CUDA_ARCH_PTX="" \ -D INSTALL_C_EXAMPLES=OFF \ -D INSTALL_TESTS=OFF ..
```

Then after this is done, compile using:

```
$   sudo make -j4
$   sudo make install
$   sudo ldconfig
```

B Description of attachments

B.1 Source Code

The source code this project is based on

B.2 Previous reports and theses

Previous reports and code this projects is based on

B.3 Datasheets

The datasheets of hardware used in this project