The Azure Kinect Manual/logbook

Created during an internship at Maastro

Francoise Broeckaert

September 7, 2021

Abstract

This document has been created to aid people who have to work with the Azure Kinect. It will shortly describe the camera itself, the installations required to use the camera (provided software from Microsoft itself and MATLAB) and the available settings. In addition, it will also provide information on the created functions (available on GitHub, see section 4) and on experiments performed with the Azure Kinect, providing both the setups as well as the results.

Note that this document purely exists to aid successors who have to work with the Azure Kinect and that there may very well be mistakes or possible improvements.

If there are any questions or comments please contact:
f.v.a.m.broeckaert@student.tue.nl / francoise.broeckaert@gmail.com

Contents

1 The Azure Kinect

The Azure Kinect is a device from Microsoft, image-wise it can provide RGB data, infrared data and depth data. In addition, it has an IMU sensor and microphones as can be seen in Figure 1.

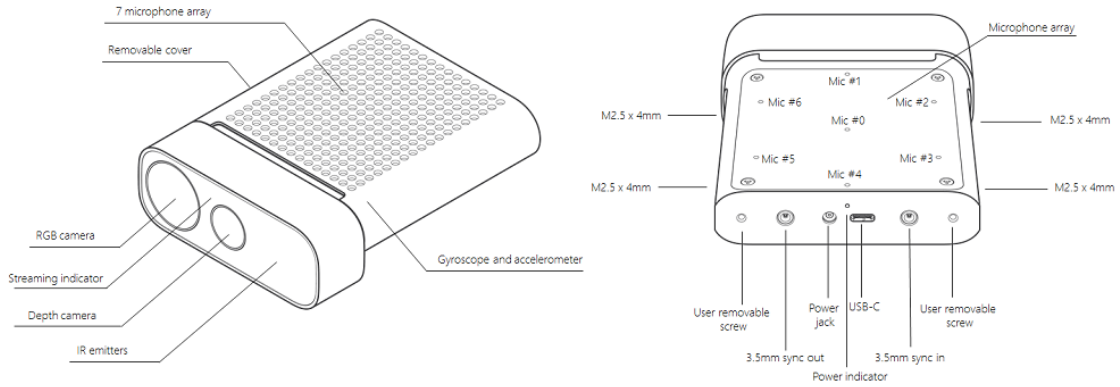


Figure 1: Hardware specifications of the Azure Kinect

The depth camera is based on a continuous-wave time-of-flight principle. Here the incorporated IR emitters send out a continuous wave which is reflected by the 3D surface and the reflected wave is detected by the depth camera. Subsequently, the phase shift is determined and based on this shift the depth is determined.

The IMU sensor integrated in the device consists of a tri-axial accelerometer and a gyroscope.

In Table 1 the device specifications are shown. It should be noted that the these are the optimum and can differ per specified setting. For instance, the frame rate has the possibilities of 0, 5, 15 and 30 fps, the best of those being 30 fps and hence this is shown in the table.

Table 1: Device specifications

Property	**Azure Kinect**
RGB Camera Resolution	3840 × 2160 px
IR Camera Resolution	1024 × 1024 px
Framerate	30 fps
Field of View IR Camera	75 × 65/120 × 120 degrees (NFOV / WFOV)
Field of View RGB Camera	90 × 59/90 × 74.3 degrees

2 Installation of the software

In order to operate the Azure Kinect the following software needs to be installed:

1. Azure Kinect sensor SDK
2. MATLAB packages:
   - Image Processing Toolbox
   - KinZ-Matlab
   - mingw 64

2.1 Azure Kinect sensor SDK

Microsoft has provided software to operate the Azure Kinect, this software can be found using the following link:
https://docs.microsoft.com/nl-nl/azure/kinect-dk/sensor-sdk-download
This link contains information about what is included in the download and the link to the actual download page. In addition, this link also shows a table of content for all documentation of the Azure Kinect by Microsoft. To download the software, scroll down and follow the instructions given. It is suggested to download the latest version (v1.4.1).

In order to view the data the Azure Kinect-Viewer needs to be installed, this is included in the download. This is a useful tool to see and explore all the possible settings, however, later in the document MATLAB will be proposed to make, save and evaluate the recordings and do other tasks.

2.2 MATLAB

To be able to use the later proposed MATLAB functions, it is necessary to have the Image Processing Toolbox and the KinZ-Matlab properly installed. In order for this to work the following needs to be done:

2.2.1 Image Processing Toolbox

There is a chance this package is already installed since it is commonly used. If this is not the case go to MATLAB, click on the Home tab and subsequently click on "Add Ons". Use the search bar and type "Image Processing Toolbox" and install the package.

2.2.2 KinZ-Matlab

The typical language to use the Azure Kinect is C++, however, it is also possible to use MATLAB since not everyone if familiar with C++ (this document therefore makes use of MATLAB instead of C++). The KinZ-Matlab toolbox has been created by Juan R. Terven & Diana M. Cordova so MATLAB can be used instead of C++. Their code as well as derivations and other code proposed in section 4 can be used to operate the Azure Kinect in MATLAB.

Downloading KinZ-Matlab The package can be downloaded in MATLAB by clicking on the Home tab after which "Add Ons" should be opened and where can be searched for KinZ-Matlab. In order to get the files separately the following link needs to be used:
https://nl.mathworks.com/matlabcentral/fileexchange/81788-kinz-matlab
here the documents can be downloaded which will be needed later on.

Install MinGW-64 MinGW-64 also needs to be installed using MATLAB Add Ons. Once this is done "mex -setup c++" needs to be typed into the command window. If this is done correctly it will say that it was successful.

Adding to path It is possible that a path still needs to be added for everything to work properly. If this is the case, the following path should also be added:
"C:\Program Files\Azure Kinect SDK v1.4.1\tools"
This can be done using the following steps:
Open "Search Windows" → type and choose "Edit the system environment variables" → click on the button "Environment Variables..." → open the "Path" under "System variables" by selecting it and clicking on "Edit" → fill in the new path and click on "OK" for all windows → restart the program or computer to let the system pick up the changes.

For a more visual guideline see:
https://www.architectryan.com/2018/03/17/add-to-the-path-on-windows-10/

Compile for Windows Lastly the "compile_for_windows.m" file will need to be run in MATLAB, this is part of the files downloaded in section 2.2.2. If done correctly the command window will again show that it has been successful.

3 Camera settings

In this section all the possible settings will be explained, these are based on the possibilities shown in the Azure Kinect Viewer. At the end of this section the default settings will be displayed. Note that the openly presented settings in the MATLAB code are less extensive.

3.1 Enable depth camera settings

Firstly, there is a choice to turn the "Enable Depth Camera" on or off. It's quite evident that turning it off will result in only having an output from the 2D color camera without the output of the infrared (IR) or depth stream. When the depth camera is enabled, there are some options for the depth configuration:

1. NFOV Binned
2. NFOV Unbinned
3. WFOV Binned
4. WFOV Unbinned
5. Passive IR

here NFOV means narrow field-of-view and WFOV means wide field-of-view. The difference between binned and unbinned is that binning combines neighboring pixels in order to improve SNR and it can increase frame rate, however, this is at the cost of spatial resolution. Passive IR means the camera does not emit IR, but only detects it. So it only detects the IR emitted by objects themselves, therefore this mode also has no field-of-light (fol) nor does it have a limited range determined by the camera.

3.2 Enable color camera settings

The second header is the "Enable Color Camera" which can be turned on or off. This section gives some more options like color configuration format (BGRA, MJPG, NV12 and YUY2) and resolution (16:9; 720p, 1080p, 1440p and 2160p, 4:3; 1536p and 3072p). Next there are the color controls allowing one to shift the exposure time, white balance, brightness, contrast, saturation, sharpness and gain. Lastly, backlight compensation can be turned on or off and the power frequency can be set to 50Hz and 60Hz.

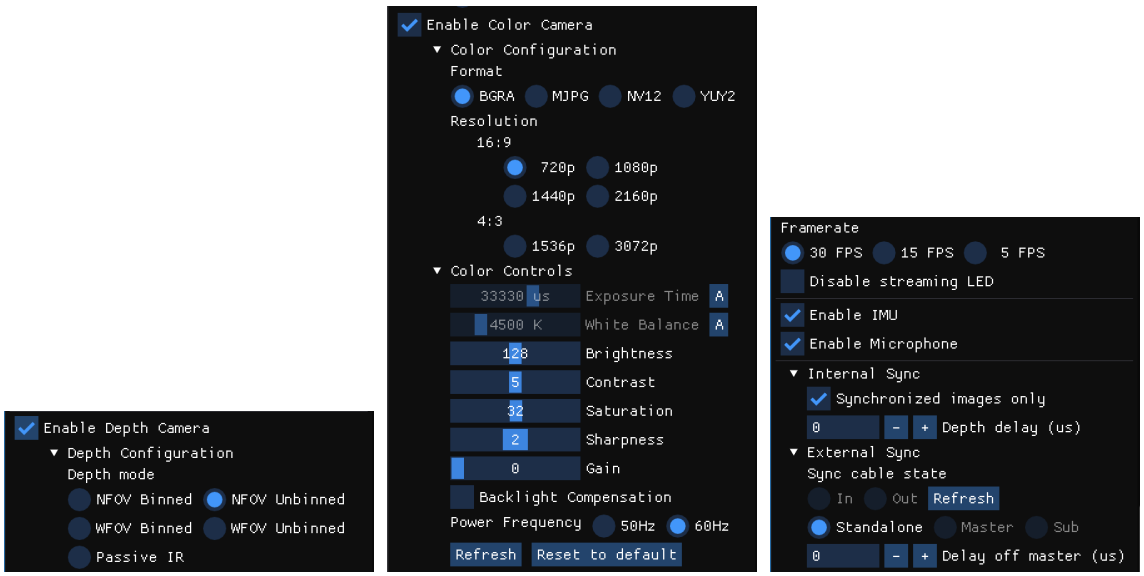
Finding the specific differences between the four color configuration formats has not been accomplished and by the naked eye a real difference was not observed. The resolution on the other hand is quite straightforward so will also not be explained, similar to the other controls such as brightness, contrast, saturation and sharpness which one can just play with to see what gives the best result for the specific recording. Gain can be used to shift color values and the backlight compensation can be used to filter out input from the effects of backlight. Lastly, there is the power frequency which describes the number of on/off times per second.

3.3 Remaining settings

The remaining settings consist of the following. There is the frame rate which can be set to 30, 15 or 5 frames per second (FPS). There is a choice to enable or disable the streaming LED, which means the light indicating whether or not the camera is active can be turned on or off. In addition, the IMU and microphone can each be enabled or disabled. The IMU stands for Inertial Measurement Unit and includes an accelerometer and a gyroscope. Lastly, there are the internal and external synchronisation options. The internal synchronization options include synchronized images only (yes or no) and a depth delay which can be set in the order of μs. The external synchronisation options relate to the merging of multiple Azure Kinects into one camera system. The camera can either be set to run standalone (so without any other cameras involved) or it can be set to either master or sub. In each system one camera needs to be set to master and the remaining camera(s) can be set to sub. Also, a delay in the order of μs can be induced relative to the master camera.

3.4 Default settings

The default settings were already turned on with the initial opening of the application. In order to have a reference of what those initial (default) settings are, Figures 2a, 2b and 2c present screenshots.



(a) Enable Depth Camera (b) Enable Color Camera (c) Remaining settings

Figure 2: Default settings for the Azure Kinect

3.5 Suggested settings

For this project the depth camera was of greatest importance and in the context of the DETERMINE project, it is assumed the patient is at a distance of 1.5-3.0 meters away. Therefore, the NFOV is advised because this ranges between 0.5-3.86 meters. It is also suggested to used unbinned data, since it is easier to bin it afterwards if needed than the other way around. In MATLAB another setting is

given, namely the out of range setting. This setting is purely for visualization as it sets the range of the color bar. It is advised to set it a slightly bit higher than the maximum distance of the object of interest. Another approach would be to use the automatic determination of the range settings, this is provided in (most of) the code on GitHub. Note that this automatic range does not yet work optimally since outliers can make a huge impact.

As MATLAB was used instead of the Azure Kinect Viewer and the focus was on the depth camera, the other settings have not been investigated and therefore no recommendations are made.

4 MATLAB functions

Instead of using the Azure Kinect Viewer, it is also possible to use MATLAB to obtain and save images.

In the following GitHub repository some functions can be found related to the Azure Kinect: https://github.com/Francoiseb99/Maastro_internship
The following functions are included in the GitHub repository:

1. ShowLiveImages
2. SaveImages
3. SaveImagesV2
4. ShowSavedImages
5. ShowSavedImagesV2
6. GetSeparateImages
7. CalcMeanDepth
8. DepthOverTime
9. ExtractData
10. MegaPlot
11. DepthOverTimeBuildUp
12. DepthOverTimeTable
13. DetStabilizationTime
14. OptimumWarmUpTime
15. ShowOrientation
16. MegaOrientationPlot
17. ShowDepthStdImage

Note that some of this code is at least to some extent based on the code generated by Juan R. Terven & Diana M. Cordova found in the demos of KinZ-Matlab.

4.1 ShowLiveImages

This function shows a live feed which can be useful if nothing needs to be saved and you just want to try something out. The input variables specify which type of footage needs to be shown, 1 being show the data and 0 being do not show the data. The possibilities are the depth data, color data and infrared data.

4.2 SaveImages

This function is used to display recordings in real time and save them as MATLAB arrays. The user has four input variables, namely filename, Depth, Color and IR. The filename is the name the arrays will be saved under. The depth, color and IR each can be turned to 1 or 0 independent of each other, 1 being show and save the particular data and 0 being do not show nor save the data.

4.3 SaveImagesV2

This function has been created to save data which can be used in the runDetection.m file. This runDetection.m file however has been created by Nick Staut and will therefore not be greatly elaborated on in this document. However, to get a small idea: the runDetection.m file needs color and IR data of a checkerboard structure in different positions which can be obtained by the SaveImagesV2, after running the runDetection.m the output from this file can be used to do run the runCalibration.m file, all this generally needs to be done only once as the values from the calibration of the camera remain the same for the particular camera. The input variables are the filename under which it needs to be saved and it needs to be specified which data needs to be shown and saved. The data which can be chosen is again Depth, Color and IR can be chosen (show: 1 or do not show: 0). However, for the code from Nick to work color and IR data need to be present.

4.4 ShowSavedImages

This function displays the footage saved by the SaveImages function. It again has four input variables, namely SavedFrames, Depth, Color and IR. In this case SavedFrames is the filename under which the data has been saved. The Depth, the Color and the IR are again a specification of which data needs to be shown, 1 being yes and 0 being no. If the data is requested by the user, but not present, the function will return 'Depth data not present in file and is therefore not shown.', or its equivalent for the color or IR data.

4.5 ShowSavedImagesV2

This function is an augmented version of ShowSavedImages and is intended to view images from the SaveImagesV2. Variables again are the SavedFrames, Depth, Color and IR. Each of which works the same as described previously.

4.6 GetSeparateImages

This function can be used to convert footage saved with the SaveImages function to separate images which will be saved at a desired location. The first input variable for this function is the SavedFrames, which is the output from the SaveImages function. Furthermore, Depth, Color and IR can again be said to either 1 (yes) or 0 (no), meaning those images are saved or not. However, due to some setbacks the destination path had to be hard coded at the end of the file and should be manually altered. In addition, the type of image can be altered and is currently set as a PNG file, this also can be found at the end of the file.

4.7 CalcMeanDepth

This function makes use of previously saved data by the SaveImages function, this data can be specified in the input variable SavedFrames. It can be used to determine the mean distance of an object over a manually selected region. When more than 1 frame is used, the mean is not limited to calculation over the area, but also over the different frames. Therefore, selecting a moving object will not give a desirable result. This function has only one input variable, namely SavedFrames which is the path to the desired footage. In addition, this function makes use of imcrop, which will let you manually select the region of interest on the first frame taken by the camera. After selecting the region the right mouse button should be used and "crop image" should clicked. Afterwards, the code will run and return the mean depth over the selected area and if applicable over the different frames.

4.8 DepthOverTime

This function can be used to evaluate the "warm-up time" of the camera. First, a region of interest can be selected manually, the average of this region will be calculated per frame. The first plot will show the mean depth of this region per frame over time. The second plot will show the mean depth of the region averaged per second (reducing the noice), as well as the standard deviation per second. The last plot will show the mean depth of the region averaged per minute (reducing the noice even more) and will also have a plot with the standard deviation per minute. Lastly, the code will display the minimum and maximum encountered mean depths and at which time they occurred, as well as the total running time in seconds, a version rewritten to minutes to make it easier to read and also the temperature difference between the start and end of measurement. It will be given as follows:
The minimum encountered average depth was ?? around ?? seconds. The maximum encountered average depth was ?? around ?? seconds. The total running time was ?? seconds, which is ?? minutes and ?? seconds. The encountered temperature difference is ?? degrees celcius.

The input for this function is the number of frames used to obtain the footage. The accompanied time can differ slightly per run, but as an example 100.000 frames takes about 55-60 minutes. Other research has also shown that the warm-up time can be around 40-50 minutes.

4.9 ExtractData

This function is used to extract data from figures created by the function DepthOverTime. The function is hard coded to extract data from specific figures. However, by changing the names manually, the code can still be used for new data. Due to the hard coding, it does not make use of any input variables.

4.10 MegaPlot

This code is not actually a function and is all hard coded. It extracts data like in the ExtractData function and uses it to combine several plots to display a measurement error over time plot for multiple runs. As it is not a function, it does not have input variables.

4.11 DepthOverTimeBuildUp

This function returns two matrices which show the build up in percentages based on specific intervals to be determined manually. One matrix bases its percentage on the average of the first interval whereas the second matrix takes the average of the first minute and determines the build up percentage with that value as starting point. The function also returns a plot of the average difference of the depth at each time point relative to the final / maximum depth. As input variables there is DepthArrays, which contains the averaged depth data per minute from multiple runs (data obtained with ExtractData funcion) and the TimeArray which contains an array of minutes for how long the run was.

4.12 DepthOverTimeTable

The DepthOverTimeTable function returns a table with the depths at certain timepoints which can be predefined using the input variable TimePoints, in addition the final time point will be included automatically. In addition to the depth at those timepoints, it will also show the difference of that depth with the maximum as well as the final depth for that run. The other two variables needed for this function are DepthArray and TimeArray, both of these can be obtained using the ExtractData function.

4.13 DetStabilizationTime

This function was created to determine the stabilization time, which initially was planned to be used as a definition for the warm-up time. It comprises of several ways to define stabilization, only differing by the specific points considered and the deviation allowed. It considers a time to be the stabilization time if the depth difference with predefined subsequent time points remains under a certain limit. In all cases, the depth difference needs to remain below the limit for the stabilization time +0, +5, +10 and +15 minutes. If desired, an addition of the maximum or final depth can be included in which case the difference with the depth at these moments should also remain below the threshold. Lastly, the threshold values can either be set manually or can be determined based on the minimum and maximum encountered value for that specific run in combination with a percentage of this difference

which can be set by the user. The input variables are again the DepthArray and TimeArray which are defined earlier. All in all, there are a lot of different ways to define stabilization time and it should be determined by the user which way is most desirable. However, in this project it was eventually chosen to take another approach and use the OptimumWarmUpTime function.

4.14 OptimumWarmUpTime

The recommended warm-up time was eventually decided to be defined as the inflection point where the slope coefficient was the same as the average slope coefficient, which was determined by taking the difference of the maximum and minimum encountered value divided by time. It was argued that everything before this point could be considered beneficial to the stabilization, whereas everything after was not worth the extra time needed. The input variables for this function are DepthArrays and TimeArray. DepthArrays is a composition of the multiple DepthArray arrays provided by ExtractData and TimeArray is just one array as these are all the same for each of the DepthArrays.

4.15 ShowOrientation

This function is used for the gyroscope evaluation experiment and shows the orientation of the camera based on output from the internal gyroscope. It makes use of summing up the output of the gyroscope, as the output is in degrees per second. It also assumes the camera starts in a horizontal position. It will show a live plot where a cube form will rotate based on the output of the camera. In addition, it will provide a plot with the cumulative output of the yaw, pitch and roll presented in degrees over time. The input variables are Duration and Filename. The duration can be set in minutes as this is an upgraded version of the nrFrames used in other functions, it will run as long as the current time is below the set duration. The second input variable is the filename under which the data can be saved.

4.16 MegaOrientationPlot

MegaOrientationPlot is similair to MegaPlot as it is simply a hard coded piece of code combining several plots from the ShowOrientation function into one big plot. It therefore also has no input variables as it is not an actual function.

4.17 ShowDepthStdImage

This function is able to provide a standard deviation map for the depth data based on multiple frames. It takes data saved by the SaveImages function and specified under the name SavedFrames. Whereas including the depth data is not optional in this function as it is mandatory, the color data can be additionally included if specified by the user by setting Color to 1.

5 Experiments log

In this section an overview of performed experiments is given. The methodology is presented as well as the actual results, however, the results are not described nor discussed. It should also be noted that the methodology section, in case available, is taken from the report.

5.1 Settings

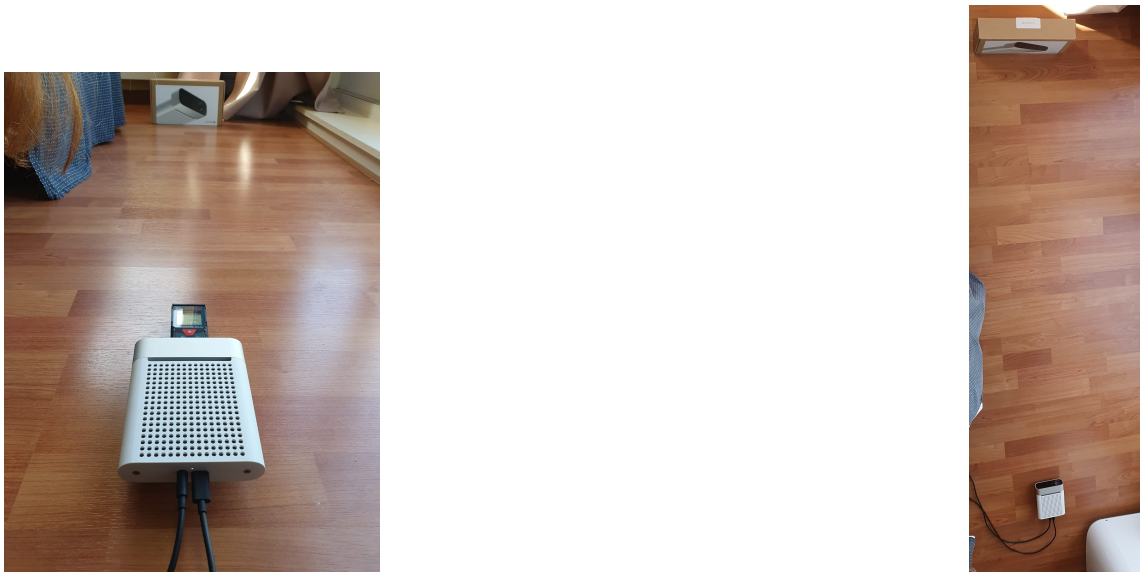
For most of the experiments the suggested settings of unbinned NFOV are used. The only exceptions are the calibration and the depth accuracy sub-experiment "Depth accuracy including flat object surfaces with different colors and at different angles for short range", where binned WFOV is used. In addition, with the sub-experiment "Depth accuracy on a flat white wall for moderate range", two modes were used, namely unbinned NFOV and binned WFOV. Overall, the ranges used in the experiments were 1.5-3.0 meters with an interval of 0.5 m. The only exception to this is the sub-experiment "Depth accuracy including flat object surfaces with different colors and at different angles for short range" where 0.45-1.0 meters were used with intervals of 0.05 m.

5.2 Calibration

In order to calibrate the camera, a 6x8 checkerboard pattern with a size of 59.5 cm x 84.2 cm was used. 200 frames with the checkerboard pattern in different positions from the camera were taken after which a single camera calibration and a stereo camera calibration between the infrared and the color camera were performed. It should be noted that for this calibration different settings were used, namely binned wide field-of-view (WFOV), as the provided calibration code only worked for these specific settings. From this calibration the lens distortion could be obtained which in turn can be used to improve the obtained images. The calibration can also be used to overlay the depth and color images.

The calibration was performed and output can be found on GitHub. However, actual follow-up steps have not been done during this internship leading to no presentable results as of yet.

5.3 Experiment: Depth estimation over time



(a) (b)

Figure 3: Pictures from the setup of the depth estimation over time experiment

This section includes the depth estimation over time experiment which is meant to display the Azure Kinects behaviour over time. First, the general experimental setup will be elaborated on, after which the determination of the advised warm-up time will be explained and a later addition of actually including the internal temperature data relative to time and depth estimation is included.

5.3.1 Depth estimation over time

In the paper of Tölgyessy *et al.* it was stated that the Azure Kinect has a warm-up time affecting its precision and this warm-up time was stated to be 40 to 60 minutes. However, since accuracy and precision are very important in the context of patient tracking during BT, it was decided to redo this experiment taking into account both aspects. This was done by determining the average distance of a manually selected area of an object and ploting the estimated distance over time (the DepthOverTime function was used, see GitHub). The object used during this experiment was a box with dimensions 15.5 cm x 29 cm (x 6 cm). However, for each run, the selected area differed as it was done manually and the aim was to just take multiple pixels in the middle of the box in order not to rely on only one pixel. Like mentioned in section 5.1, the desired distance is between 1.5 and 3.0 meters, therefore the depth over time has been established for a range of distances between these two with an interval of 0.5 meters. To evaluate the repeatability of this experiment, three runs have been performed for the 1.5 m distance. To ensure accuracy of placement of the object opposed to the camera, a laser based measurement tool was used. This way not only precision over time could be evaluated, but also the accuracy. To make sure the camera was not warmed-up beforehand, the experiment was done at the beginning of the day before the camera was used for other purposes. A 1.000.000 frames were taken, translating to approximately 9 to 10 hours of footage. Such a big time window was chosen to ensure a good overview and not conclude anything

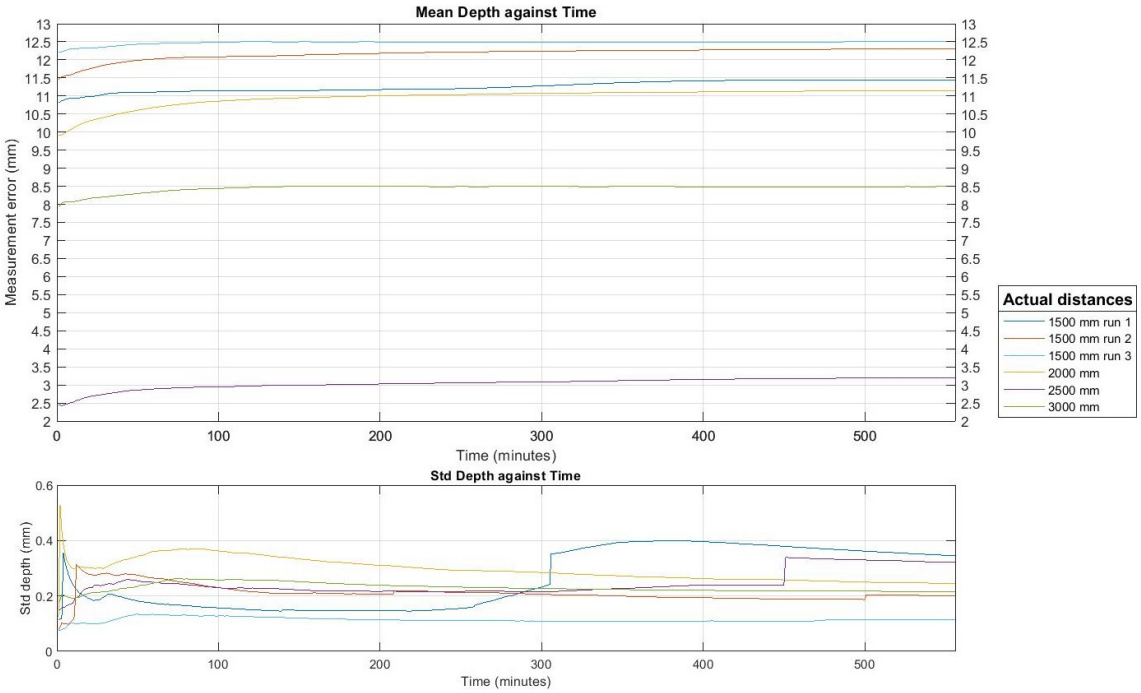


Figure 4: This figure shows the results from the depth estimation over time experiment. The results in the first subplot are normalized by subtracting the actual distance from the measured distance with the Azure Kinect resulting in the measurement error. The actual distance per line is given in the legend.

based on a possible intermediate plateau. Pictures of the experimental setup are shown in Figure 3.

The results are plotted in Figure 4. For more information and discussion points on these results, see the official report (which can be found on GitHub).

5.3.2 Determination warm-up time

To determine a warm-up time for clinical use, a few steps were taken. First, for each of the measurements (minus 1500 mm run 3, as this one was not yet obtained) the difference between the actual encountered depth at each time point and the maximum and final depth were taken. Secondly, this was plotted and the average slope coefficient was determined. Lastly, the average slope coefficient was matched to the time point in the actual graph having the same slope coefficient, this will be called the inflection point. Based on the curve of the graph it can be stated that everything before this time point has a steeper slope coefficient. Therefore everything before this point is considered to be beneficial in the warm-up time whereas everything after does not give enough added value opposed to the extra time needed to be invested. To determine this optimum warm-up time a MATLAB function was used, namely the OptimumWarmUpTime function (see GitHub).

Results from this sub-experiment are shown in Figure 5. Again, for more information, see the official report.

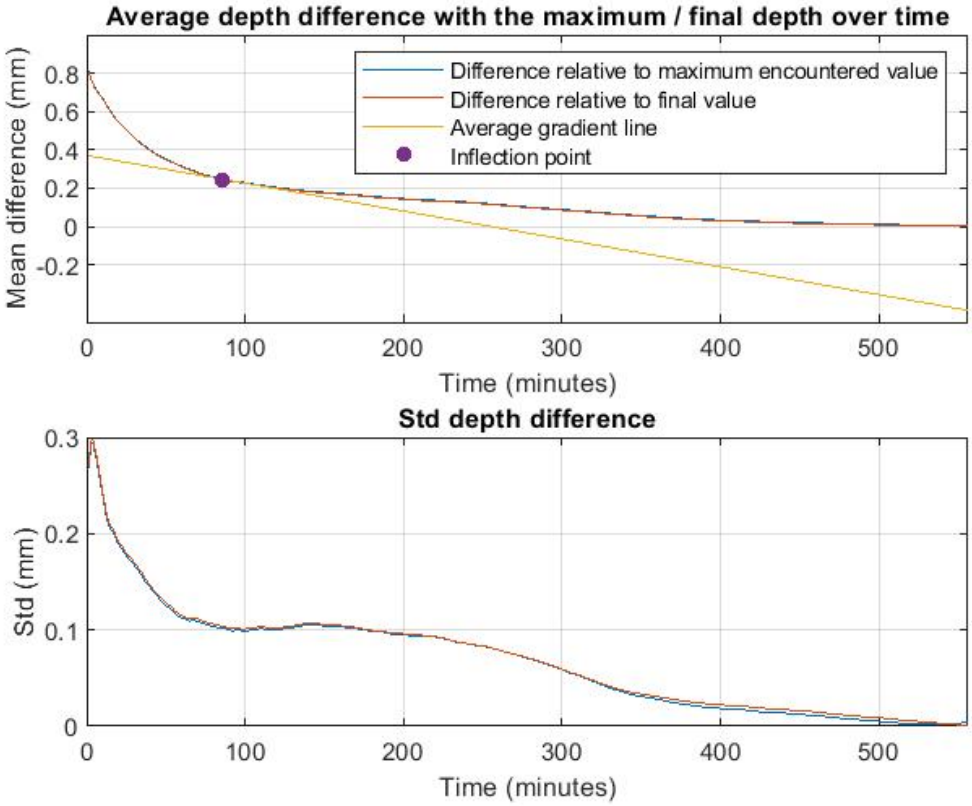
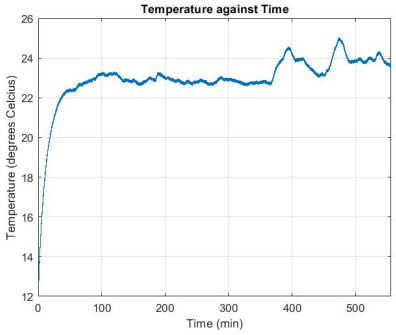
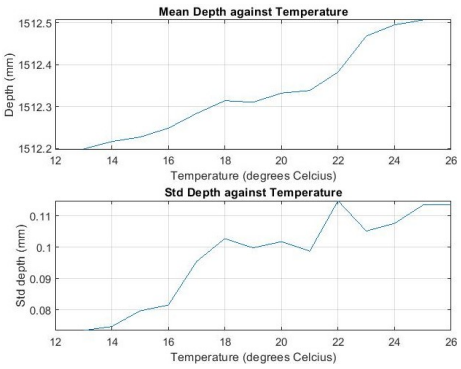


Figure 5: This figure shows two plots. The first plot shows the average depth difference based on the maximum / final encountered depth value, an average gradient line and the point at which the gradient is equal to the average gradient (line). The second plot shows the standard deviation of the above plot.



(a) Temperature against time



(b) Mean depth against temperature and its standard deviation

Figure 6: These figures show the relation between temperature and time as well as the relation between temperature and depth estimation.

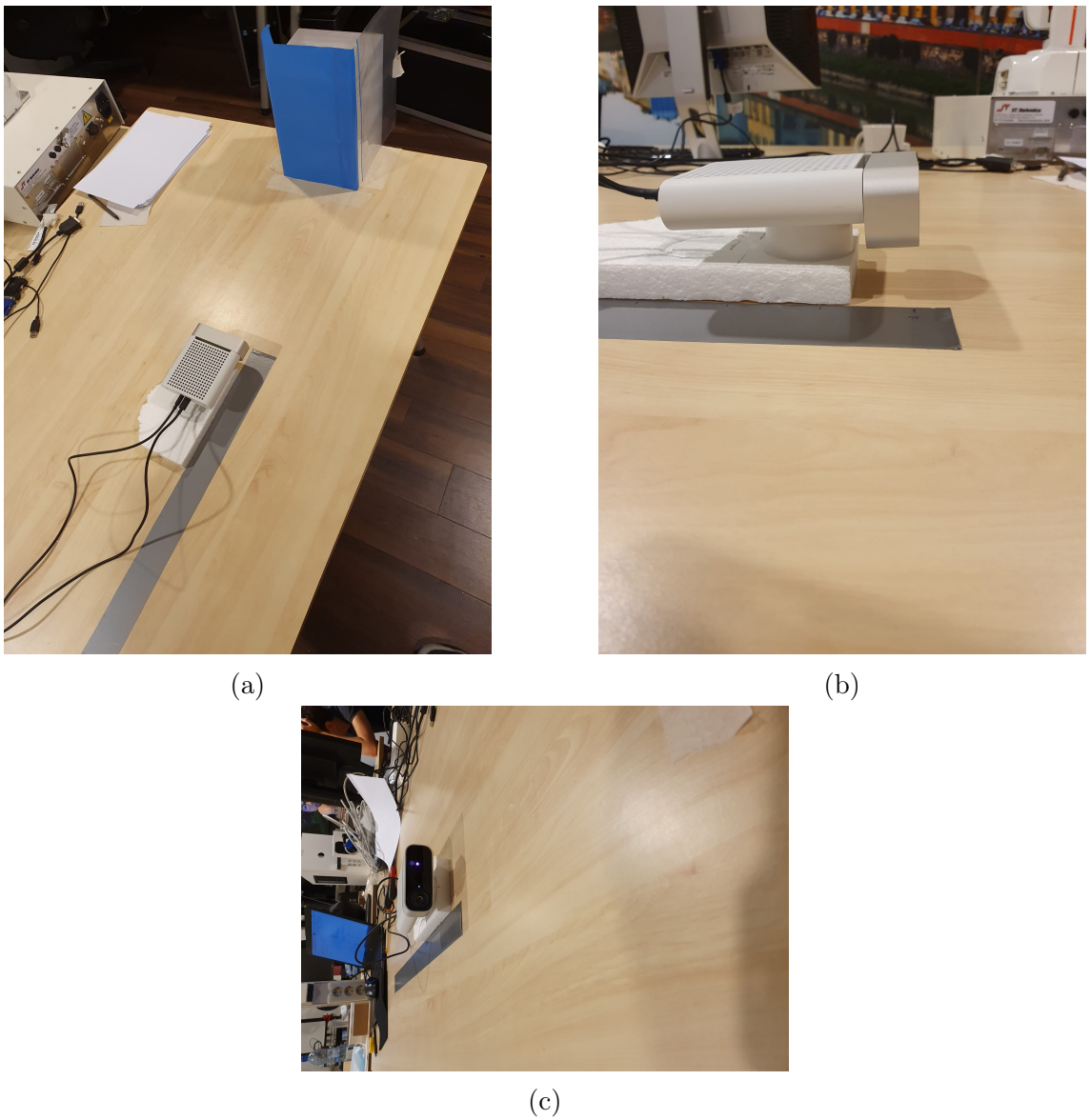
5.3.3 Temperature

For 1500 mm run 3, the internally measured temperature of the camera was taken during each iteration. This allowed for plots to be made relating time and temperature as well as temperature and depth to evaluate how this relation holds. This

experiment was performed indoors at normal room temperature.

The results are shown in Figure 6 and discussed in the report.

5.4 Experiment: Depth accuracy



(a) (b)

(c)

Figure 7: Pictures from the setup of the gyroscope evaluation experiment

5.4.1 Depth accuracy including flat object surfaces with different colors and at different angles for short range

In order to determine the accuracy of the Azure Kinect, a series of measurements were taken in different setups. Firstly, a PMME block was used and a color sheet was attached to this, see Figure 7a. (Note that a PMME block is not necessary, just something to hold up the sheet in a straight repeatable position.) Three different colors were used to evaluate the effect of different colors on depth accuracy, the colors light blue, dark blue and red were used. Secondly, the two placements of the block were marked so that this experiment could be repeated if necessary. Two angles were used, 0 degrees (parallel to the camera lens) and 45 degrees. Thirdly, a piece of duct tape was placed perpendicular to the block on which the different distances

could be written down. Measurements were taken from 450 mm to 1000 mm with an interval of 50 mm, these distances were written down on the tape in order to ensure repeatability. The front of the camera was aligned with the notations on the tape, see Figure 7b. Lastly, a piece of styrofoam was used to place the camera on, this piece of styrofoam had a flat side which could be placed against the duct tape with distances from the block to ensure the angle of the camera was the same for each measurement. In addition, on both the styrofoam and the camera standard a little stripe should be added with a pencil so these can be aligned each time as can be seen in Figure 7c (this is necessary because the standard of the camera is round).

The function SaveImages was used to record and save each of the combinations for 25 frames. Subsequently, the created file was called upon by the function CalcMeanDepth, which calculated the mean depth of a manually selected area over the 25 frames. It should be noted that due to different positioning and accuracy from the camera as well as the human component in the manual selection, the selected area could differ for each setup. This could explain some slight deviation between the measured distance by hand and by camera. In addition, it should be noted that even if the surface was flat, a point at the top of the object could still have a little deviation relative to a point in the middle of the object since the distance to the camera would differ slightly.

Distance measured (mm)	**Distance camera single point (mm)**	**Distance camera average (mm)**	**Difference between distance measured and distance camera averaged (mm)**
450	463	460.97	-10.97
500	515	513.00	-13.00
550	566	564.61	-14.61
600	618	616.32	-16.32
650	669	665.93	-15.93
700	718	717.11	-17.11
750	763	763.72	-13.72
800	811	809.52	-9.52
850	857	854.22	-4.22
900	900	897.54	2.46
950	945	941.47	8.53
1000	986	983.17	16.83

Table 2: Settings: object color is light blue, angle is 0

Results of these experiments can be found in Tables 2, 3, 4, 5, 6 and 7.

5.4.2 Depth accuracy on a flat white wall for moderate range

The second sub-experiment made use of a flat white wall and was meant to determine the measurement error for two modes, namely binned WFOV and unbinned NFOV. The second mode was included to evaluate potential differences. A white wall with landmarks to determine the center in the depth image was used, for pictures of this setup see Figure 8. The center and the alignment of the camera were verified by two cross line lasers. After placing the camera at the respective distances with the help of a laser measurement tool and verifying its position respective to the lasers, 25 frames were taken in each mode. Afterwards, the center in the depth image was determined manually using the landmarks. The coordinates of the center were used to determine the pixel value throughout all frames from which the mean and standard deviation

Distance measured (mm)	**Distance camera single point (mm)**	**Distance camera average (mm)**	**Difference between distance measured and distance camera averaged (mm)**
450	451	441.96	8.04
500	499	502.22	-2.22
550	557	558.15	-8.15
600	609	619.47	-19.47
650	665	662.42	-12.42
700	708	719.87	-19.87
750	761	768.82	-18.82
800	804	813.78	-13.78
850	846	859.74	-9.74
900	888	895.14	4.86
950	929	933.06	16.94
1000	962	969.59	30.41

Table 3: Settings: object color is light blue, angle is 45

Distance measured (mm)	**Distance camera single point (mm)**	**Distance camera average (mm)**	**Difference between distance measured and distance camera averaged (mm)**
450	462	461.89	-11.89
500	512	514.19	-14.19
550	567	569.71	-19.71
600	618	623.05	-23.05
650	666	671.09	-21.09
700	722	723.94	-23.94
750	768	769.06	-19.06
800	816	814.15	-14.15
850	862	858.98	-8.98
900	907	901.64	-1.64
950	951	944.44	5.56
1000	995	986.51	13.49

Table 4: Settings: object color is red, angle is 0

Distance measured (mm)	**Distance camera single point (mm)**	**Distance camera average (mm)**	**Difference between distance measured and distance camera averaged (mm)**
450	462	417.77	32.23
500	523	476.98	23.02
550	599	539.33	10.67
600	676	605.48	-5.48
650	714	705.55	-55.55
700	757	742.78	-42.78
750	790	780.12	-30.12
800	827	817.76	-17.76
850	868	858.81	-8.81
900	895	889.66	10.34
950	923	922.95	27.05
1000	945	958.21	41.79

Table 5: Settings: object color is red, angle is 45

could be calculated. Lastly, the difference of the average estimations by the camera and the set distances were taken to determine the measurement error.

The measurement errors of both modes can be found in Figure 9 and are discussed and compared to the original paper from Tölgyessy *et al.* in the report.

Distance measured (mm)	Distance camera single point (mm)	Distance camera average (mm)	Difference between distance measured and distance camera averaged (mm)
450	456	453.90	-3.90
500	507	508.80	-8.80
550	561	564.66	-14.66
600	612	617.34	-17.34
650	661	667.26	-17.26
700	715	718.47	-18.47
750	765	766.43	-16.43
800	806	806.57	-6.57
850	853	850.59	-0.59
900	899	896.04	3.96
950	941	935.92	14.08
1000	986	980.71	19.29

Table 6: Settings: object color is dark blue, angle is 0

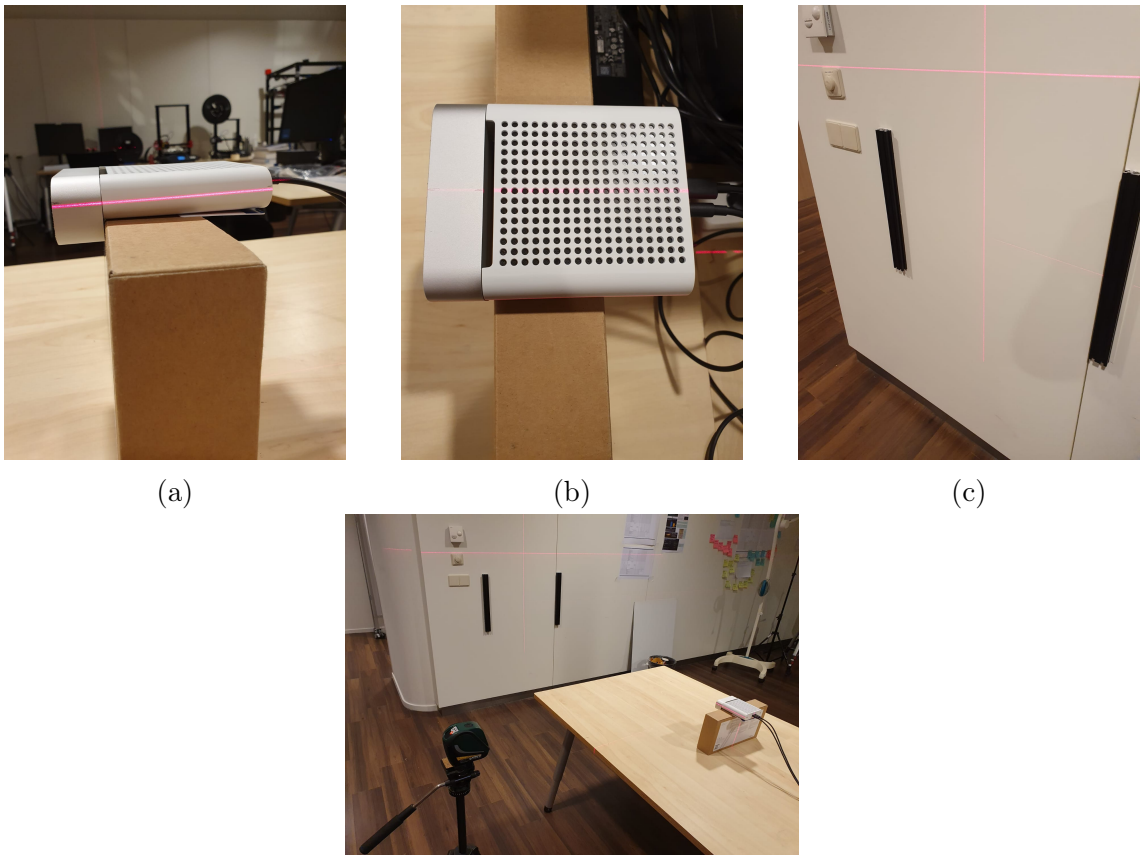
Distance measured (mm)	Distance camera single point (mm)	Distance camera average (mm)	Difference between distance measured and distance camera averaged (mm)
450	438	436.00	14.00
500	507	489.22	10.78
550	561	554.56	-4.56
600	626	587.80	12.20
650	644	624.80	25.20
700	675	650.33	49.67
750	723	778.86	-28.86
800	0	0.00	800.00
850	0	0.00	850.00
900	0	0.00	900.00
950	0	0.00	950.00
1000	0	0.00	1000.00

Table 7: Settings: object color is dark blue, angle is 45

5.4.3 Depth accuracy including flat and gradient object surfaces with different colors for moderate range

In this third sub-experiment two types of surfaces were used as well as two different colors for each. Firstly, there was the flat surface with the colors blue and red. Secondly, two 3D printed phantom heads were use, one being light of color and the other dark. The objects were placed at the moderate distances, for the head phantoms this distance was calculated from the nose, with the help of a laser measurement tool. In addition a cross line laser was used to make sure the objects, especially the heads, were placed at the correct position. For each setup, 25 frames were taken and saved for evaluation where the average estimated distance of a single pixel given by the camera could be compared to the distance measured by the laser measurement tool. For an illustration of the setup used during this experiment, see Figure 10.

The results of this experiment are presented in Table 8 and Figure 11



(a) (b) (c)

(d)

Figure 8: Pictures from the setup of the depth accuracy on a flat white wall for moderate range experiment

Distance (mm)	**Estimated distance mean ± std (mm)**			
	Blue block	*Red block*	*Dark head*	*Light head*
1500	1506 ±0.79	1509 ±1.02	0 ±0.00	1518 ±0.91
2000	2004 ±1.01	2007 ±1.08	0 ±0.00	2019 ±1.21
2500	2502 ±1.68	2500 ±1.29	0 ±0.00	2518 ±1.32
3000	3005 ±1.80	3022 ±1.99	0 ±0.00	3031 ±1.56

Table 8: These are the results from the depth accuracy experiment for flat and gradient object surfaces with different colors. For the two blocks, the distance from the depth camera to the middle of the block is used. For the two heads, the distance from the depth camera to the nose are used.

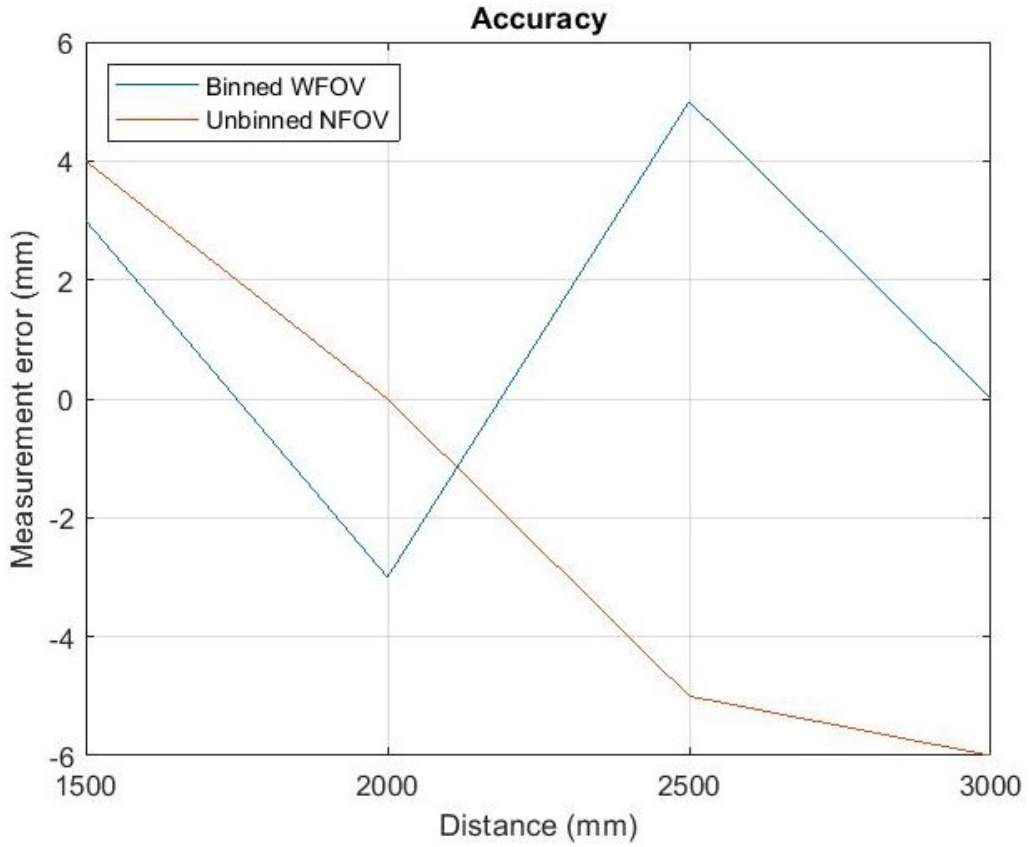
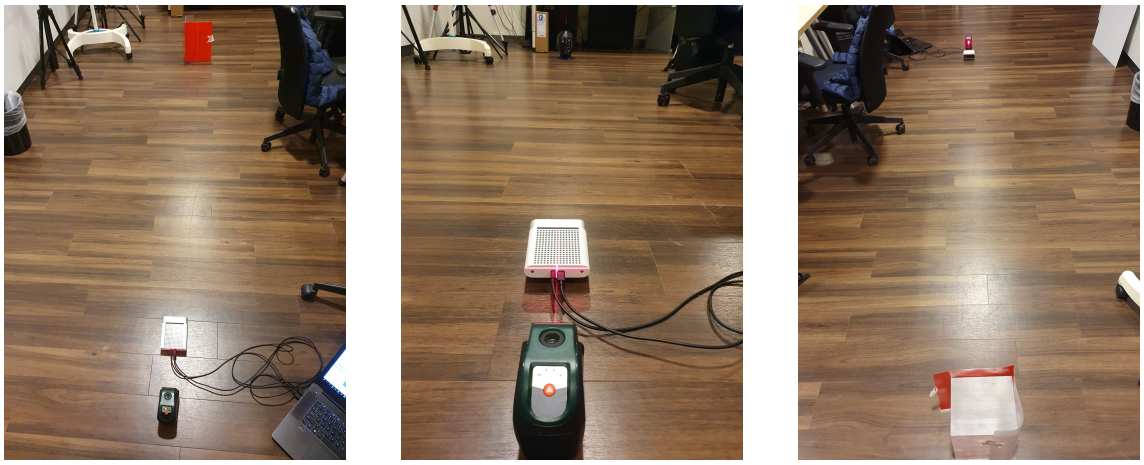
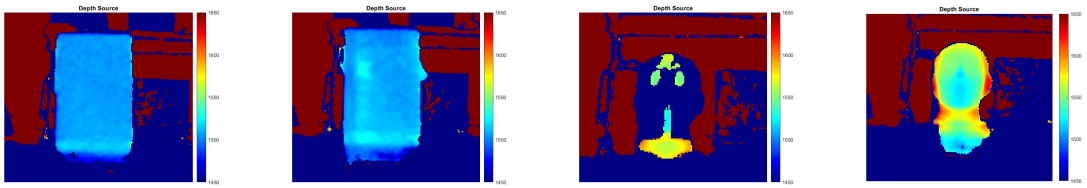


Figure 9: This figure shows the measurement errors for two modes at different distances. The standard deviation was 1 mm for each point.



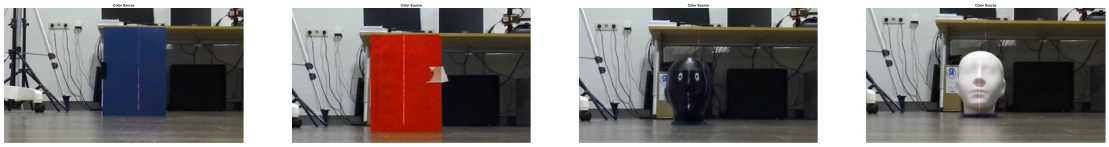
(a) (b) (c)

Figure 10: Pictures from the setup of the depth accuracy including flat and gradient object surfaces with different colors for moderate range experiment



(a) Blue block depth map (b) Red block depth map (c) Dark head depth map (d) Light head depth map

(e) Blue block std map (f) Red block std map (g) Dark head std map (h) Light head std map



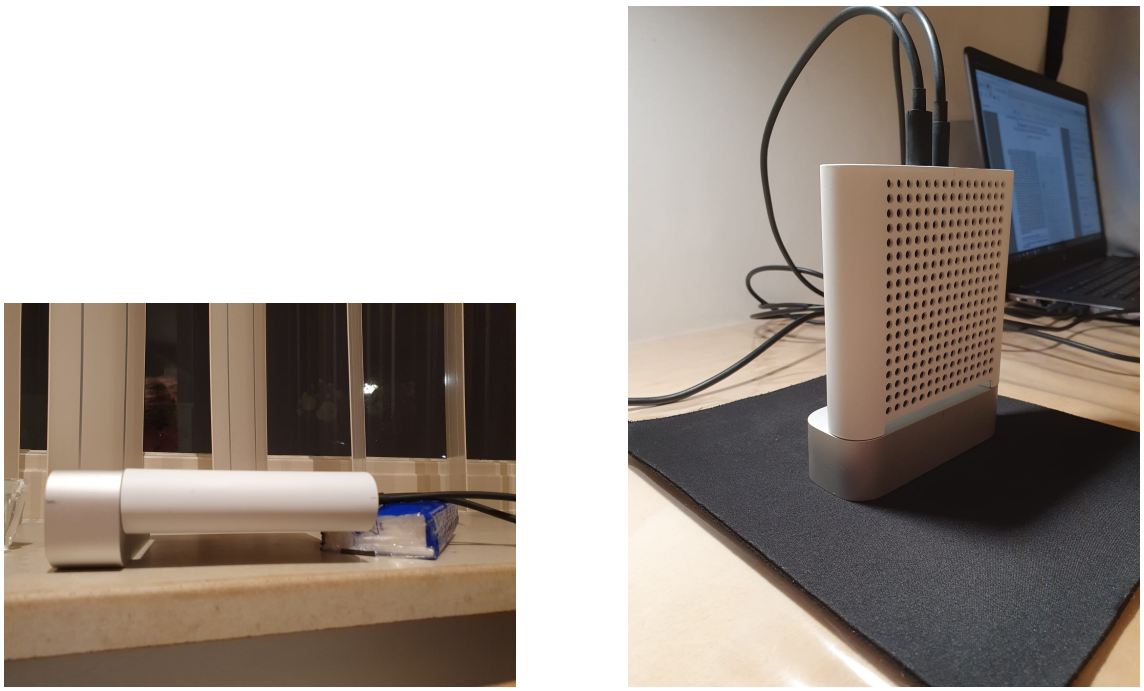
(i) Blue block color image (j) Red block color image (k) Dark head color image (l) Light head color image

Figure 11: The first row shows depth images taken during the depth accuracy experiment at a distance of 1500 mm. To be able to visually spot the nuances, the color range has been set to 1450 mm - 1650 mm. The second row visualizes a standard deviation map based on the 25 frames taken per run. The third row shows the RGB images of the objects.

5.5 Gyroscope

Lastly, the internal gyroscope was evaluated to see if it could be used for depth corrections in a later stage. In order to evaluate the stability of the gyroscope, three runs were performed. During the first two runs the camera was placed horizontally on a flat surface and during the third run the camera was placed vertically on a flat surface, see Figure 12. In these positions the camera was activated and the gyroscope output was taken and plotted cumulatively over time. This was done for approximately 250 minutes to see if the output would remain constant or would show a certain drift.

The cumulative output from the pitch, yaw and roll of the different runs is shown in Figure 13 and are discussed in the report.



(a) (b)

Figure 12: Pictures from the setup of the gyroscope evaluation experiment

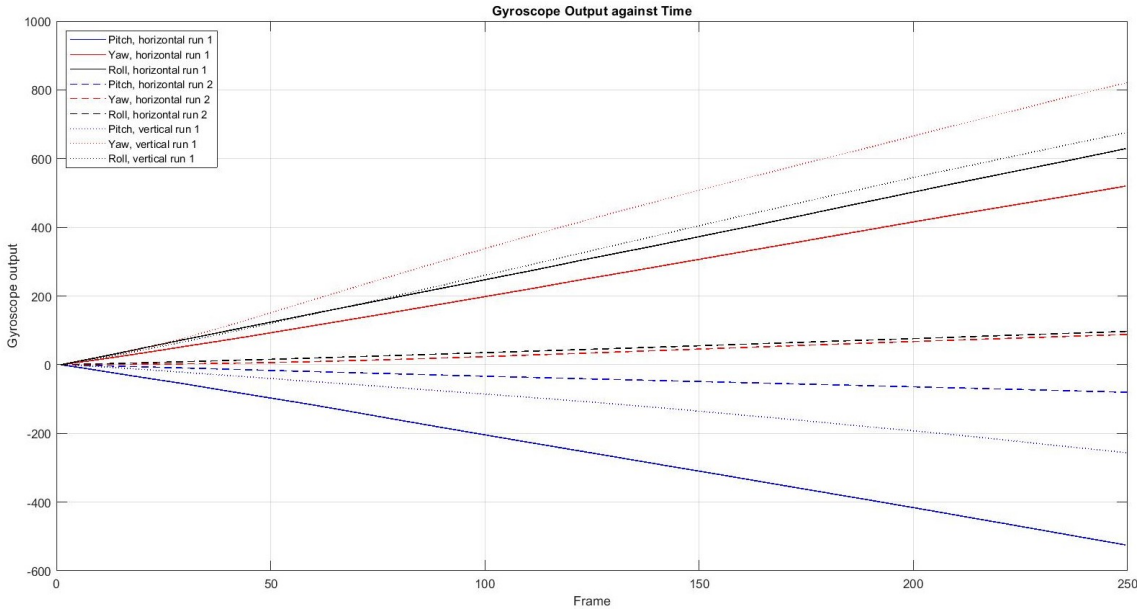


Figure 13: This figure shows a summation of the output of the internal gyroscope for $\sim$250 minutes for each of the three runs.