File: P21311 Camera Documentation

MSD Team: Mind Controlled Wheelchair

Owner: Cheyenne Dailey - cjd9104

Date: 2/8/2021

## 

[**Purpose**](#_7oxr6cbfl4ln) **2**

[Overview](#_tpvqs165bztb) 2

[Design Choice](#_8q2qubqj4tu1) 2

[**Resources**](#_7csfwhhmmd9i) **3**

[**Installation**](#_wy2zop2q66gz) **3**

[Ubuntu Image](#_bzrzlltl3i6a) 3

[Librealsense and RealSense Viewer](#_x6nhdmowdu2t) 3

[**Code Samples**](#_9advuv9bmf25) **4**

[**D435 Depth Camera**](#_8cm913y0mywg) **5**

[Specs](#_n5ogqo87rnw) 5

[Intel RealSense Viewer](#_qqdbe37b6d93) 6

[Observations](#_rciyimsyh7oi) 7

[**T265 Tracking Camera**](#_1vj0cdeaaelm) **7**

[Specs](#_utnkmhymu3tu) 7

[Intel RealSense Viewer](#_cyh8mwvi2sd3) 8

[Observations](#_qgxt5quxo9yn) 9

# Purpose

## Overview

In order to operate in a semi-autonomous state, a method of viewing the surrounding environment along with mapping the environment was necessary for safe movement and navigation. To accomplish this, a camera system that could detect objects and map a location was desired. A recommended method of implementing this was to use Simultaneous Localization and Mapping Algorithms (SLAM) Mapping.

## Design Choice

Our customer recommended looking into [Intel RealSense Depth and Tracking Camera Bundle](https://store.intelrealsense.com/buy-intel-realsense-depth-and-tracking-bundle.html). This bundle included the D435 Depth Camera with the ability to detect an object’s distance away from the camera. It also included the T265 Tracking Camera which has built in V-SLAM technology that would run directly on the device. Upon further research, an open source library was found, librealsense, that integrates with Intel RealSense devices, allowing for easier integration as well as an application for viewing camera output and example code for basic camera tasks (distance reading, pose reading, multi camera interfacing, recording, etc.). Another feature available with using the Intel RealSense SDK is the provided wrappers. By default, most of the SDK code was written in C++, along with the examples. The wrapper option allows for the choice of which language to write in; ROS, Python, Matlab, OpenCV, etc. These factors led us to believe that using these cameras was the best course of action.

The second decision that had to be made was what type of processing system would be used. Initially a Raspberry Pi 4 was selected due to its low power consumption, size, and cost. After much testing it appeared that there was always an error or some aspect of the camera functionality that was non-operable. With much research it was discovered that the 265 Tracking Camera was known for these bug issues and Intel had discontinued support for the device. After scouring different support forums, it was found that the Raspberry Pi was the likely cause of the issue, and that using a laptop would be a better option to use the cameras.

The options with a laptop were to use a Windows OS or a Linux OS. After much consideration, it was decided that using a Linux OS would be the best course of action. Linux allowed for easier compilation of code and examples. Using a team members laptop initially, a Linux OS, Ubuntu 18.04.5, was set up and used. Upon testing, all the previous issues faced with the Raspberry Pi 4 were gone, allowing for seamless multi-camera integration.

# Resources

1. ~~Raspberry Pi 4~~
   1. ~~Power Cable~~
   2. ~~Micro HDMI to HDMI~~
2. ~~Monitor~~
3. ~~Mouse/Keyboard~~
4. Laptop with Linux OS - Ubuntu 18.04
5. Intel Realsense Depth Camera D435
6. Intel Realsense Tracking Camera T265

# Installation

## Ubuntu Image

The laptop used was booted with a Linux OS (ubuntu-18.04.5-desktop-amd64.iso).

## Librealsense and RealSense Viewer

The first course of action is to clone the github repository for the Intel Realsense Library, librealsense, using the following command.

>> git clone <https://github.com/IntelRealSense/librealsense.git>

>> cd librealsense/

The instructions that were followed for installing the Librealsense library and RealSense Viewer can be found [here](https://dev.intelrealsense.com/docs/compiling-librealsense-for-linux-ubuntu-guide). For simplicity, follow the instructions below.

Setup Preliminaries

>> sudo apt-get update && sudo apt-get upgrade && sudo apt-get dist-upgrade

>> sudo apt-get install git libssl-dev libusb-1.0-0-dev pkg-config libgtk-3-dev

>> sudo apt-get install libglfw3-dev libgl1-mesa-dev libglu1-mesa-de

>> sudo cp config/99-realsense-libusb.rules /etc/udev/rules.d/

>> sudo udevadm control --reload-rules && udevadm trigger

Patch to the kernel

>> ./scripts/patch-realsense-ubuntu-lts.sh

Build Librealsense

>> sudo add-apt-repository ppa:ubuntu-toolchain-r/test

>> sudo apt-get update

>> sudo apt-get install gcc-5 g++-5

>> sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-5 60 --slave /usr/bin/g++ g++ /usr/bin/g++-5

>> sudo update-alternatives --set gcc "/usr/bin/gcc-5"

>> mkdir build && cd build

>> cmake ../

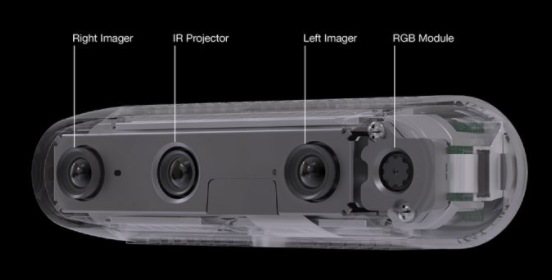
>> sudo make uninstall && make clean && make && sudo make install

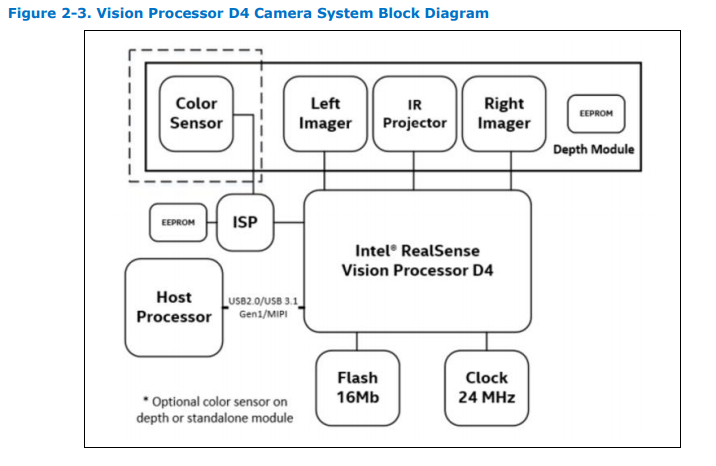
# Code Samples

Online code samples can be found for different camera functionality, for both individual use and multi camera use. Intel RealSense provides sample code within the librealsense library [here](https://github.com/IntelRealSense/librealsense/tree/master/examples) and descriptions of what those examples do along with which camera(s) are used can be found [here](https://dev.intelrealsense.com/docs/code-samples).

# D435 Depth Camera

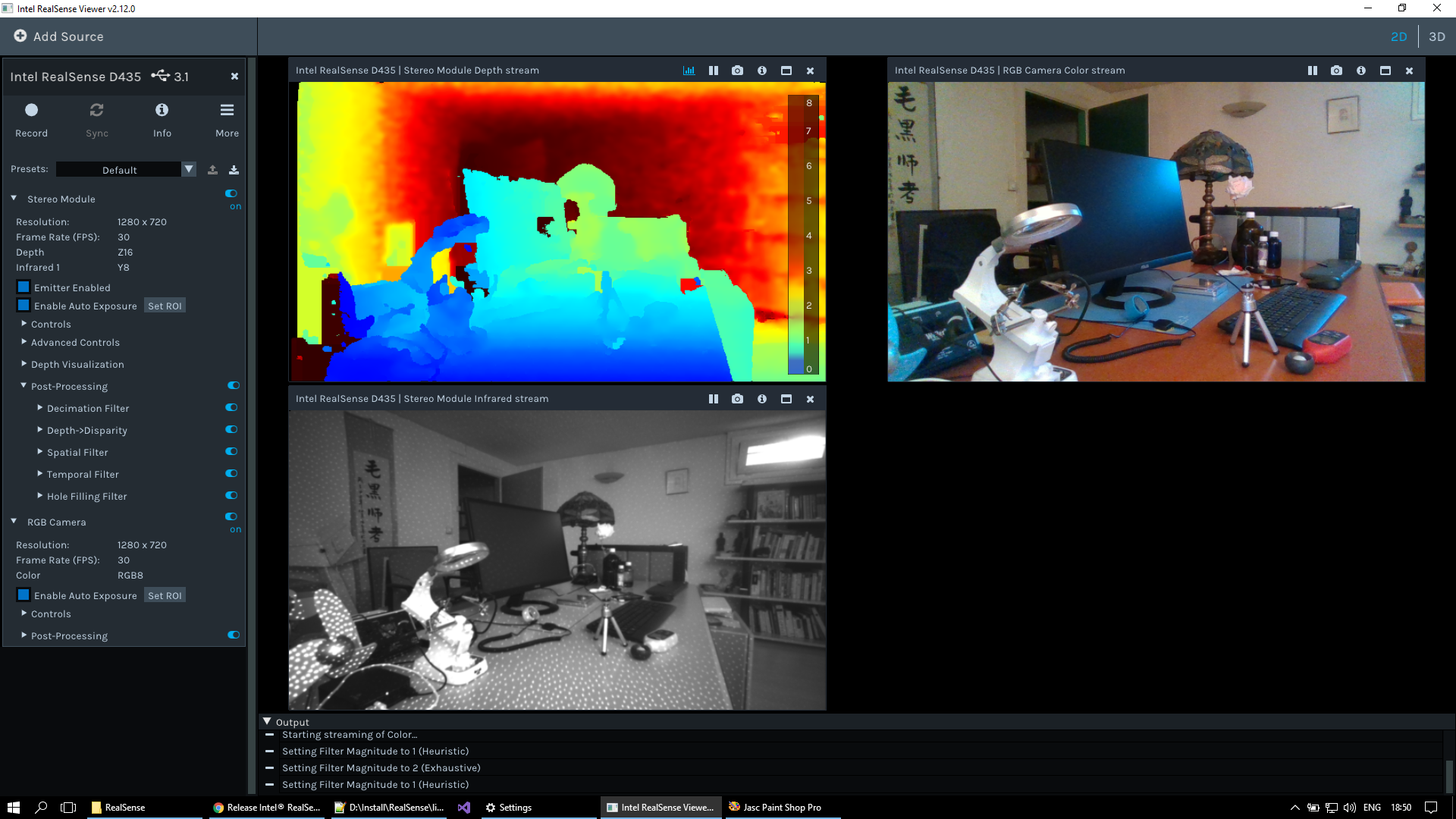
## Specs





|  |  |
| --- | --- |
| Depth Technology | Active IR Stereo |
| Main Intel RealSense Component | - Intel RealSense Vision Processor D4  - Intel RealSense Module D430 + RGB Camera |
| Depth FOV | 85.2𝆩 x 58𝆩 x 94𝆩 (+/- 3𝆩) |
| Depth Stream Output Resolution | Up to 1280 x 720 |
| Depth Stream Output Frame Rate | Up to 90 fps |
| Minimum Depth Distance (Min-z) | 0.1m |
| Maximum Range | ~10 Meters (varies depending on settings) |
| RGB Sensor Resolution and Frame Rate | 1920 x 1080 at 30 Frames per second |
| RGB Sensor FOV | 69.4𝆩 x 42.5𝆩 x 77𝆩 (+/- 3𝆩) |
| Dimensions | 90 mm x 25 mm x 25 mm |
| USB | USB-C\* 3.1 Gen 1 |
| Mounting | - One 1/4-20 UNC thread mounting point  - Two M3 thread mounting points |

## Intel RealSense Viewer



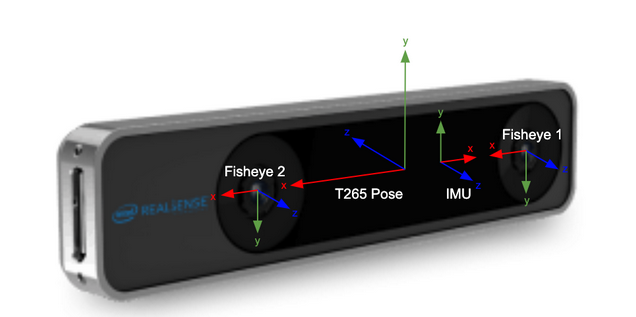
There are three viewing options with the Intel RealSense Viewer 2D rendering for the depth camera. The upper left hand corner the Stereo Module Depth Stream, using color coding to distinguish the distance each object is being registered at. Shades of blue represent closer objects while shades of red denotes objects further away. The upper right image shows the RGB Color Camera Stream. This is a realtime colored image of what is seen by the depth camera. The last view, the bottom left corner, is the Stereo Module Infrared Stream. This stream utilizes dots to provide a rendering of how dense the object it is viewing appears. The closer and more solid the object, the more dots used to distinguish the object’s location.

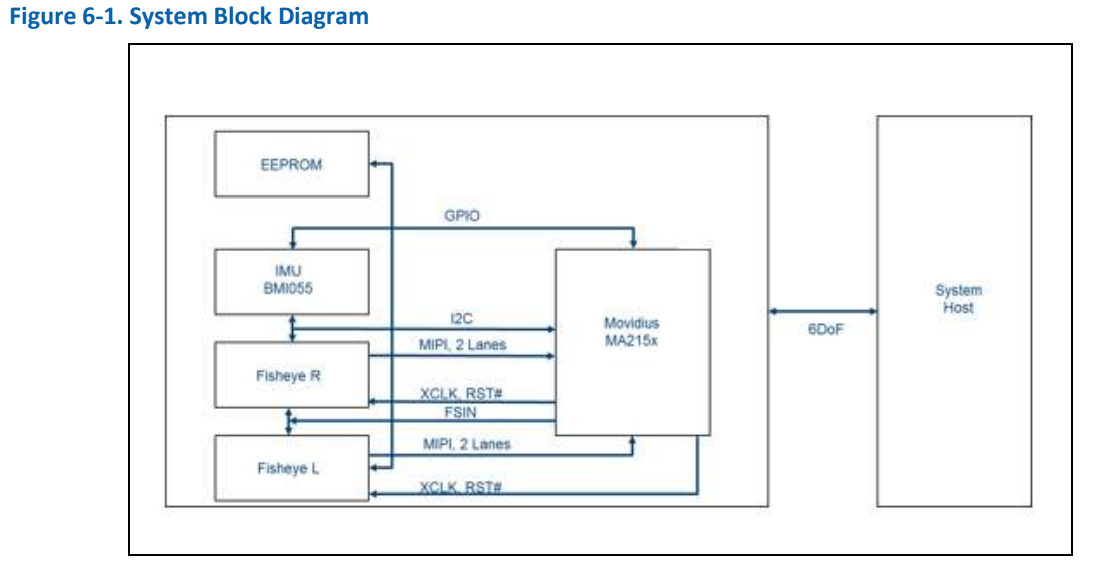
## Observations

1. Accurate depth readings tend to be within the range of 0.2m and 2m generally

# T265 Tracking Camera

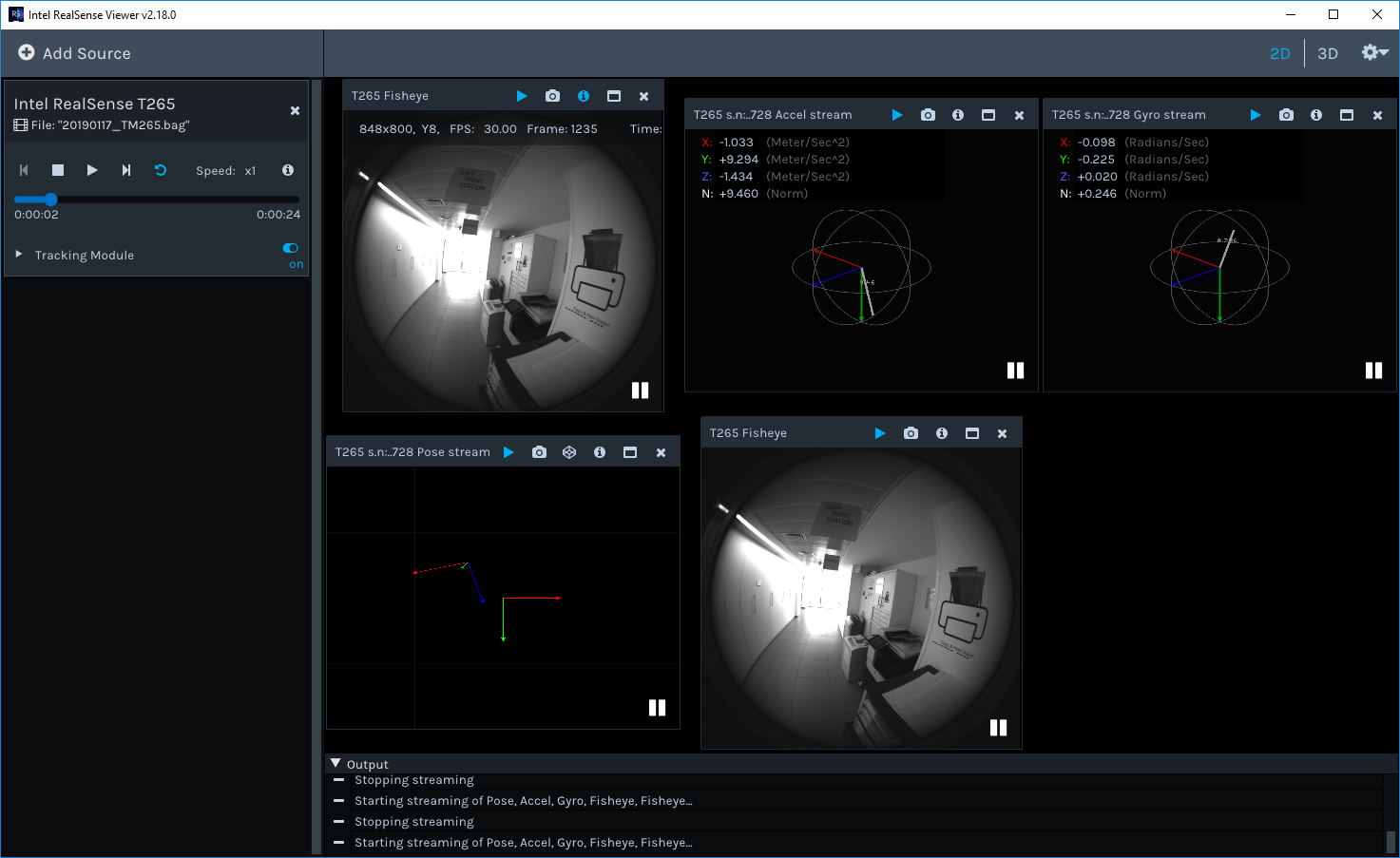
## Specs





|  |  |
| --- | --- |
| SLAM | V-SLAM : High Precision Visual Inertial Odometry Simultaneous Localization and Mapping algorithms |
| VPU | Intel Movidius Myriad 2.0 VPU : Visual Processing Unit optimized to run V-SLAM at low power |
| FOV | Two Fisheye Lenses with combined 163+5° FOV: Two OV9282 imagers with fisheye lenses for a combined, close to hemispherical 163±5° field of view for robust tracking even with fast motion. |
| Inertial Measurement Unit (IMU) | BM1055 IMU : allows for accurate measurement of rotation and acceleration of the device, to feed into the V-SLAM algorithms |
| USB | USB 3.1 Gen 1 Micro B : USB 2.0 and USB 3.1 supported for either pure pose data or a combination of pose and images |
| Dimensions | 108 mm x 24.5 mm x 12.5 mm |
| Mounting | 2 x M3 0.5 mm pitch mounting sockets |

## Intel RealSense Viewer



There are five different viewing windows for the Tracking Camera. Two, the top left and the bottom middle, represent the view from the fish-eye lens equipped with the camera. The top middle is the Accel Stream which displays the direction of acceleration that the tracking camera is moving in. The top right is the Gyro Stream which displays the tilt of the camera from its starting position. Lastly, the bottom left is the Pose Stream. This stream tracks the directions of change from the starting position of the tracking camera.

## Observations

NOTE: If the tracking camera is not registering with the example code, open the realsense-viewer. Make sure that the tracking camera is connected. Select “MORE” then select “Hardware Reset”. After that please try to run the example again

# Object Detection Design

The design implemented for our object detection relies solely on the depth camera, D435. The D435 supplies a frame of pixels, the height 480 pixels and the width 848 pixels, each corresponding to a specific distance. To determine an accurate measurement of the distance of an object, an average of a group of pixels is taken. The provided depth sample code from Intel RealSense takes an average of the entire frame, returning the overall, average depth distance that is being viewed by the camera.

For the purpose of this project, the single average was insufficient to accurately report where and how far away objects were. To remedy this, the frame was first split into three columns, providing left, middle and right.

|  |  |  |
| --- | --- | --- |
| Left | Middle | Right |

This was tested with different objects in different locations and the results proved to be lacking in the accuracy that we were striving to achieve. There were instances where the floor being registered skewed the whole column’s values which was not ideal. To remedy this, it was decided to try splitting the frame into a 4x4 grid, with 16 unique quadrants. This would allow for more specific recognition of where objects were and if they needed to be avoided. The resulting grid is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Q0 | Q1 | Q2 | Q3 |
| Q4 | Q5 | Q6 | Q7 |
| Q8 | Q9 | Q10 | Q11 |
| Q12 | Q13 | Q14 | Q15 |

This larger breakdown of the frame allowed for a better idea of where the object detected was in relation to the chair. The lower quadrants, Q12 - Q15 were ignored in testing with the emulator as it was registering the floor. This was modified of the occasion when the camera was angled high enough to avoid registering the floor the entire time.

The quadrants allowed for the identification if an object was in the direct path of the chair or if it was simply registered by the camera. The division also allowed for the recognition of if the chair needed to move to the left or the right to avoid a detected object, and how hard/far