DRAFT DOCUMENT – THIS INITIAL RELEASE IS MEANT TO FILL A NEED TO EXPORT LARGE QUANTITIES OF TANGO DATA FROM THE TABLETS. IT IS NOT FOR THE FAINT OF HEART. IT IS THE MOST INVASIVE BLABBERMOUTH WITH RESPECT TO YOUR PERSONAL PRIVACY YOU HAVE EVER MET. IN 30 MINUTES IT CAN FLATTEN YOUR BATTERY. UNLESS YOU HAVE A TANGO DEVICE AND A RATIONAL IDEA OF WHAT TO DO WITH TENS OF THOUSANDS OF POINT CLOUDS, POSES, AND PICTURES, YOU REALLY DON'T WANT THIS APP.

User Guide

1 Introduction

1.1 Scope and Purpose

This explains how to use the Tango Tricorder, an Android™ application used to manage bulk acquisition and cloud upload of the five key data streams that originate on a Tango device. These streams are;

- **Session** Marks off the boundaries of each individual Tango service connect. Tango coordinates are only valid within the scope of a single session if not localized.
- **Location** This is the high resolution feed from the GPS and gives the GPS coordinates of the device. This is used to project relative Tango coordinates into the real world.
- **Pose** This is the report of the devices orientation and relative location in space relative to motion across the session/localized area.
- **Points** This is a block of three dimensional points representing the surface locations identified by Tango.
- **Picture** This is a JPEG of controllable quality taken with the same camera that the Tango is using to generate the surface points.

Needless to say, you need a Tango enabled device to use this software, but anyone can consume the data it generates. It is designed to work directly with RESTful cloud services, or to store data locally when the cloud services are not available. It can also directly export raw and json files for the five separate streams to ease integration efforts and support alternate data consumption strategies.

1.2 Process Overview

The Tango Tricorder is realized as an application that uses the left hand navigation drawer UI paradigm, i.e. you navigate to major functional areas by dragging/flicking a sliding drawer out of the left side of the screen.

These major functional area are;

Management This is where a previously captured local file can be uploaded, or in desperation, previously captured local files can be exported for consumption by external systems.

Scanner This is the whizzy bit! Through this you see a live camera view and the point cloud rendered directly on it. This view has been optimized to support the needs of the user during data capture.

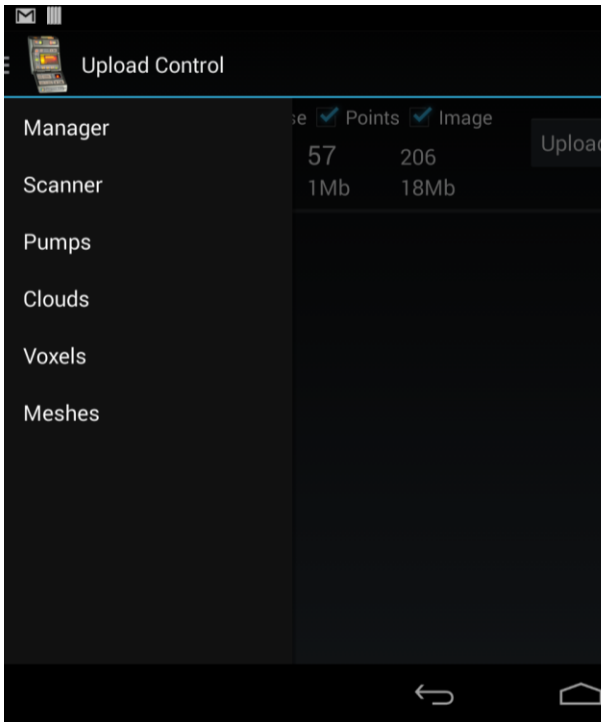
Pumps This is under active development. It shows information about the state of the major internal systems that manage the flow of data. An example of a key metric reported is the miss rate, i.e. how many incoming raw data elements of a particular type had to be tossed because the system was backed up with data.

Points This will appear next – it will provide a means to interact with large sets of point clouds, both on the local device and the cloud. This will allow for some assessment of data quality, sampling drift, etc before committing to a significant data collection, or at least informing a strategy to the best way to do the collection.

Voxel Space - Minecraft with better resolution – treat the Tango data for what it really is – voxel hits.

Meshes - Nirvana - the system is now generating mesh surfaces in real time from sensed Tango data. This will require a lot of real time cloud support and shows little promise in an offline environment.

2 Navigation and Indicators



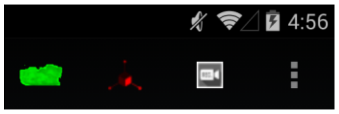
2.1 Navigation Drawer

The interface uses the standard left hand navigation drawer, which can be accessed by dragging from the left edge of the screen. The choices presented correspond to the major functional areas of the system. Selecting any item will activate the UI for that area, which will appear on the main display. The navigation drawer will then automatically close.

It should be noted that when the Tango Scanner UI element is active, the navigation drawer can exhibit considerable sluggishness when requested to open. It may even take several attempts.[1]

The last three choices do not represent currently available functionality.

2.2 Indicator Buttons



There are four icons displayed at the top right on the Action bar. From left to right, they represent cloud services, tango localization, data recording, and "*miscellaneous other stuff*" respectively. All of these icons respond to taps to perform specific functions.

Cloud Connectivity This icon is green when cloud services are available and will be used, and red when they are not available or will not be used. Tapping the cloud icon can be used to selectively enable or disable cloud services, if they are available. This is handy when the connection is too sketchy to trust with large data volumes.

Tango is Localized Green when Tango has reported that it is returning localized coordinates relative to the active area definition file. Red when the coordinates are not localized. Tapping on this icon will reset the Tango motion tracking system, resetting the Tango start of service origin if it is not localized. Resets are is not recommended under the premise that already questionable data is not improved by fiddling with it further. Unless, of course, Tango is obviously drunk (again) in which case their use is

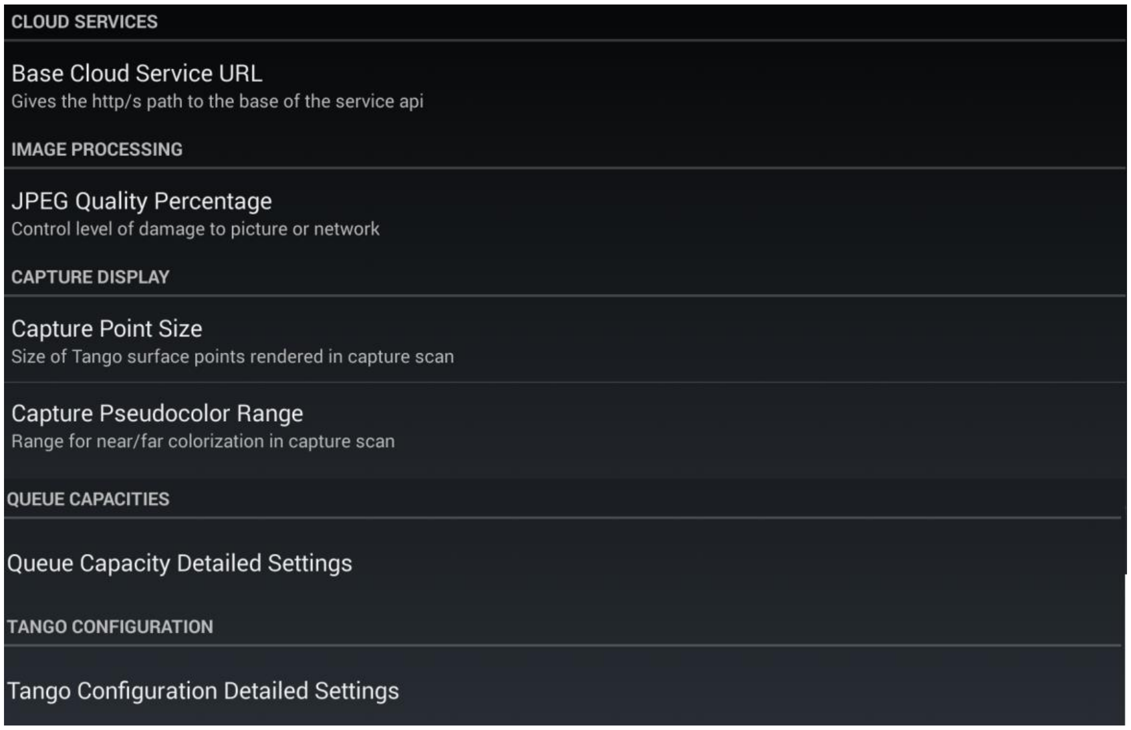
[1] It's probably unfair to single out the navigation drawer, as everything but the scanner display can be sluggish.

quite beneficial. Note that you need to let Tango look around for a bit in order for it to localize.

Recording in Progress When the system is actively recording, the recording icon is red, and when it is not recording the icon is gray Video recording ~~can~~should only be used from the scanner display, and is turned on and off by tapping the icon. Recording directly to the cloud means dealing with buffering to handle lags in the network, and recording to memory means the wholescale enthusiastic depletion of memory resources. Local memory recording is cumulative until the files are purged, i.e. the system can capture data in multiple sessions before transmittion.

The final icon provides access to secondary functions. Currently this is an information box (both useful and legal, you work out the Venn diagram) about the software and the preferences page.

3 Preferences



CLOUD SERVICES

Base Cloud Service URL
Gives the http/s path to the base of the service api

IMAGE PROCESSING

JPEG Quality Percentage
Control level of damage to picture or network

CAPTURE DISPLAY

Capture Point Size
Size of Tango surface points rendered in capture scan

Capture Pseudocolor Range
Range for near/far colorization in capture scan

QUEUE CAPACITIES

Queue Capacity Detailed Settings

TANGO CONFIGURATION

Tango Configuration Detailed Settings

The preferences screen allows you to set the cloud services connection, and fiddle with certain operating characteristics of the system.

3.1 Cloud Services/Base Cloud Service URL

The foremost element is the base cloud service URL, to which are appended the various RESTful endpoints, such as api/tango/points. Set this URL to the base address for your REST services such that adding 'api/tango/*entity*' will yield valid endpoints.

If you have no REST services then you can ignore this, but you are limited to capturing data locally and then exporting the local files off of your machine.

3.2 Image Processing/JPEG quality

The JPEG quality setting is used to control (roughly) how large the JPEG images are. Use this to control memory and bandwidth consumption. Note that high levels of accuracy in any given picture are not the goal, the goal is lots and lots and lots of pictures. This value must be between 1 and 100.

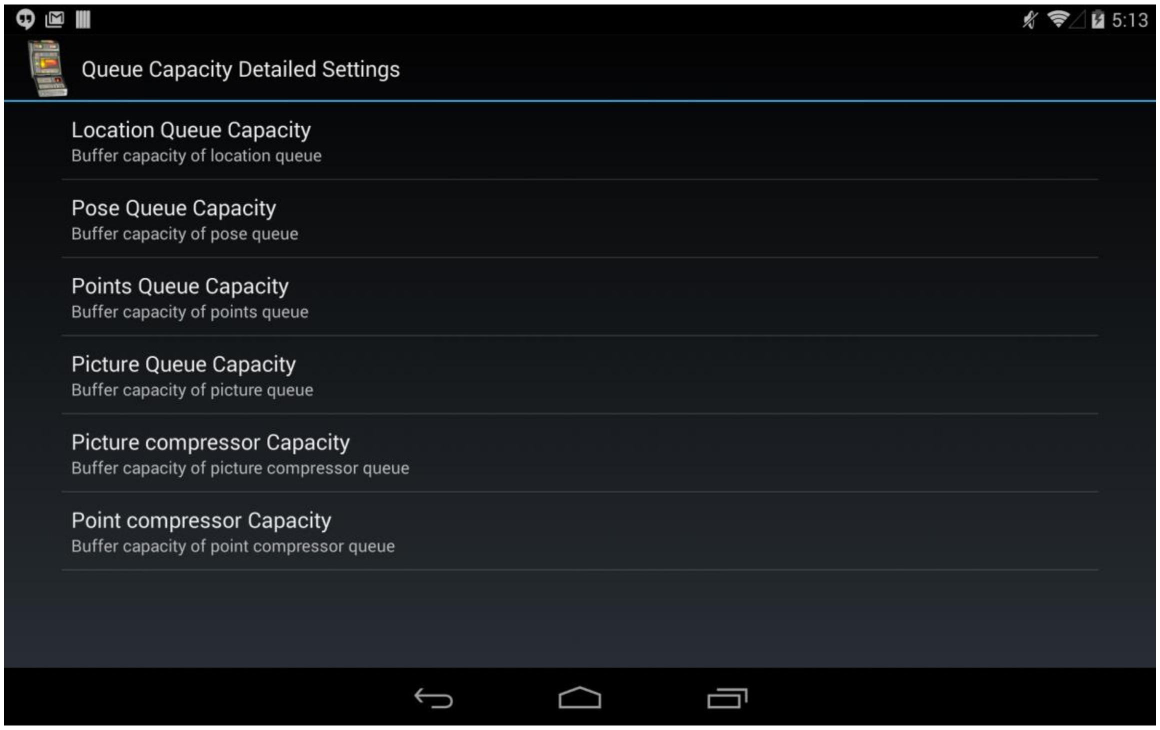
3.3 Capture Display/Capture Point Size

Used to set the default capture point size, which determines how small or fat and blobby the individual cloud pixels are when drawn. This can be interactively modified by drags on the scanner window, as explained later on

3.4 Capture Display/Capture Pseudocolor Range

Used to set the default capture distance range, which determines how cloud pixels are colorized basted on their distance from Tango. This can be interactively modified by drags on the scanner window, as explained later on.

3.5 Queue Capacity Detailed Settings


5:13

Queue Capacity Detailed Settings

Location Queue Capacity
Buffer capacity of location queue

Pose Queue Capacity
Buffer capacity of pose queue

Points Queue Capacity
Buffer capacity of points queue

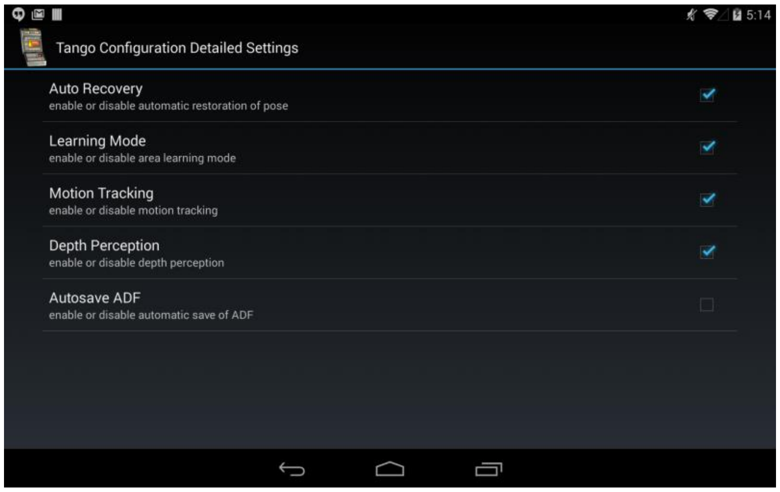
Picture Queue Capacity
Buffer capacity of picture queue

Picture compressor Capacity
Buffer capacity of picture compressor queue

Point compressor Capacity
Buffer capacity of point compressor queue

The queue capacity detailed settings are all for setting capacities of the internal data queues. Larger capacities do allow for compensating for momentary glitches, but if your data rate is higher than your channel rate, then you are going to lose stuff after the queues all back up. You'll see KPI information about this on the pump display. You can find information on strategies for managing pump buffer allocations in the section on the pump monitor display.

3.6 Tango Configuration Detailed Settings


5:14

Tango Configuration Detailed Settings

Auto Recovery ✓
enable or disable automatic restoration of pose

Learning Mode ✓
enable or disable area learning mode

Motion Tracking ✓
enable or disable motion tracking

Depth Perception ✓
enable or disable depth perception

Autosave ADF ☐
enable or disable automatic save of ADF

The Tango Configuration Detailed Settings are used to configure Tango operating characteristics. In most cases they don't need to be changed.

3.6.1 Auto Recovery

Tango will automatically attempt to reestablish pose if it stumbles if this is enabled. If it is not enabled then you have to manually trigger reestablishing the pose by tapping on the Tango icon in the action bar.

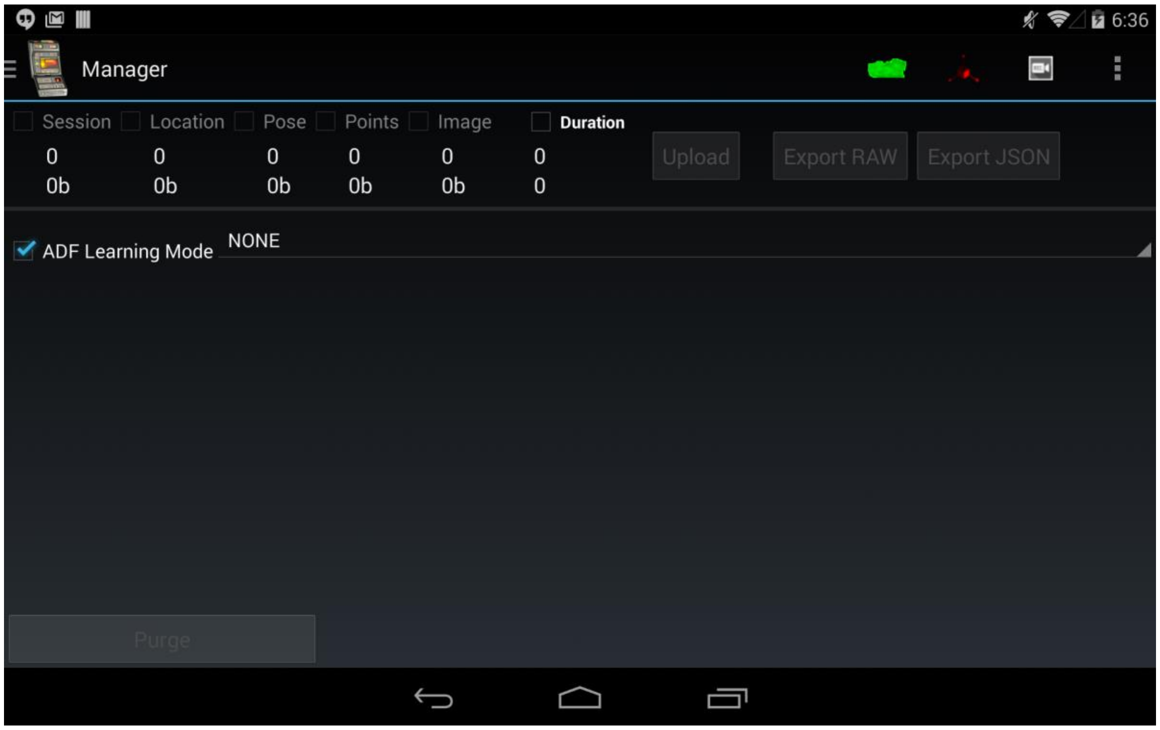
3.6.2 Learning Mode
When enabled, the system automatically adds to its knowledge base in the ADF files as they are used.

3.6.3 Motion Tracking
Needed to enable pose information return, unwise to disable

3.6.4 Depth Perception
Needed to generate point cloud, unwise to disable

3.6.5 Autosave ADF
System will automatically save updated ADF files on disconnect from Tango.

4 Manager Screen



Session	Location	Pose	Points	Image	Duration
0	0	0	0	0	0
0b	0b	0b	0b	0b	0

✓ ADF Learning Mode NONE

At the top left is the display of what is currently in the local file store. The top count defines the number of items and the bottom count the size of the file, e.g. there are 197 images in a 17Mb file.

4.1 Uploading Previously Captured Data

The "Upload" button to the right is used to upload the files to the RESTful services. This utilizes exactly the same mechanism as real time uploads, but it does it in a patient fashion and doesn't start tossing data when the cloud server is slow. For that reason, in less optimal communications environments you may want to consider locally capturing the data and then batch uploading it later.

4.2 Purging Uploaded or Unwanted Data

The “Purge” button at the lower left is used to delete the local file store. You should press this button whenever you complete an upload or just don’t want the data. Note that uploads are not transacted, i.e. if you get halfway through an upload and your battery dies, you’re going to end up doing it all over again.

4.3 Exporting Data

Data is always exported to the public external media download folder. Using ES File Explorer or an equivalent, you will find the files in Home/Download/Tango Tricorder. Note that the filenames are the same every time, i.e. if you do two exports and don’t collect the files from the first, they will be overwritten by the second. Note also that the device automatically maps the first virtual SD card back on top of internal storage, care needs to be taken to not run out of storage. It is advisable to use some tool to immediately copy these files off onto the SD card. Personally, I like ES File Explorer.

4.3.1 Exporting Raw Data

The Export RAW button is used to export the raw binary data files to the public external media download file. The raw binary files will be copied to the export directory. It’s a good idea to immediately purge the local files afterward and then copy the exported files off the machine, especially if they are quite large. Which they often tend to be.

4.3.2 Exporting JSON Data

You may also export the data in JSON format. Needless to say, these files are larger, but the point clouds and images only swell up due to base 64 encoding, the core data remains compressed. Nevertheless, heed the warnings with respect to RAW data and get the files off the device, or at least on to the SD card.

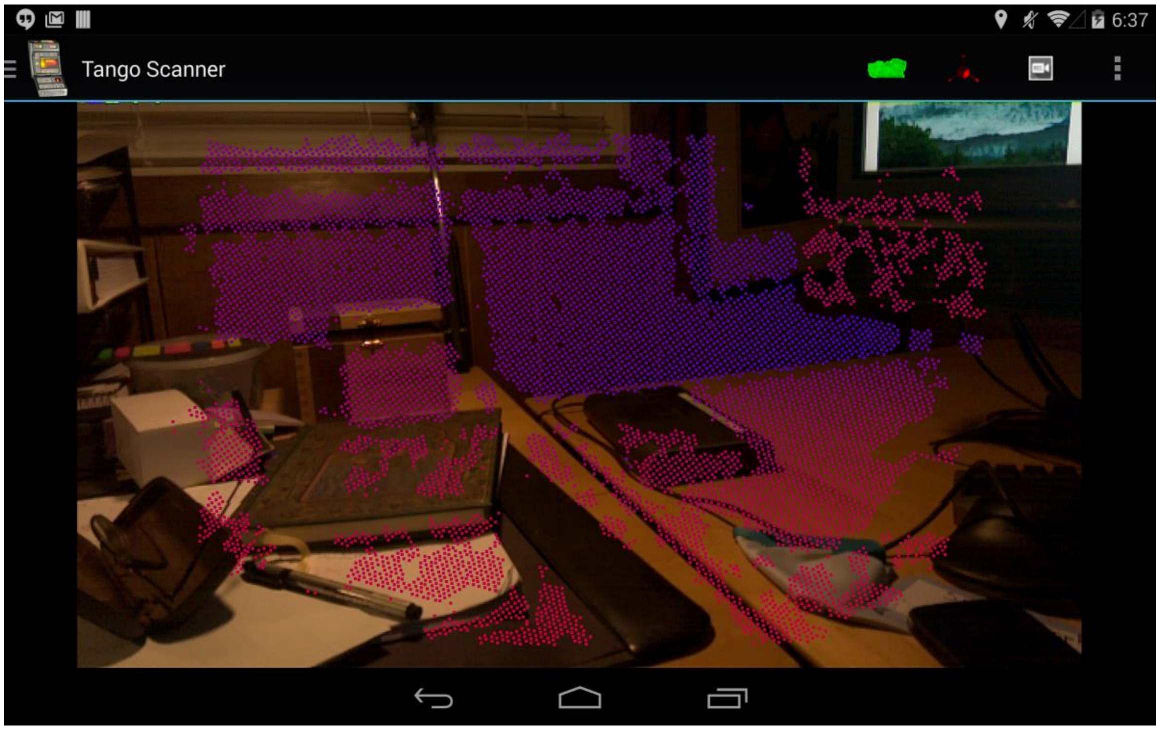
4.4 ADF Management

In my experience ADF files are vital to getting higher quality pose information, i.e. reducing overall drift. While the use of ADF doesn’t remove all drift, it makes it less of a problem. So please use them, unless you like incoherent data.

The ADF Learning mode checkbox is checked when the system is expected to save new knowledge to selected ADF files or to a newly learned environment. In the latter case the Tango API doesn’t provide a means to name the new environment and often fails, therefore I recommend you use the Tango Explorer to do all initial area training.

The spinner to the right of the checkbox allows you to select from the available ADF files. You should make a selection from this spinner **prior** to entering the scanner display. Selecting **“NONE”** is equivalent to deliberately electing not to use an ADF file and run a significant probability of lurching off a cliff.

4.5 Scanner



This is the fun bit. This was originally developed as a mashup of the augmented reality and point cloud samples from Google, and since then has sprouted a life of its own. It should be noted that pretty much everything involving direct interaction with Tango happens in the C++ native layer, Java is used primarily for UI, data management, and comms.

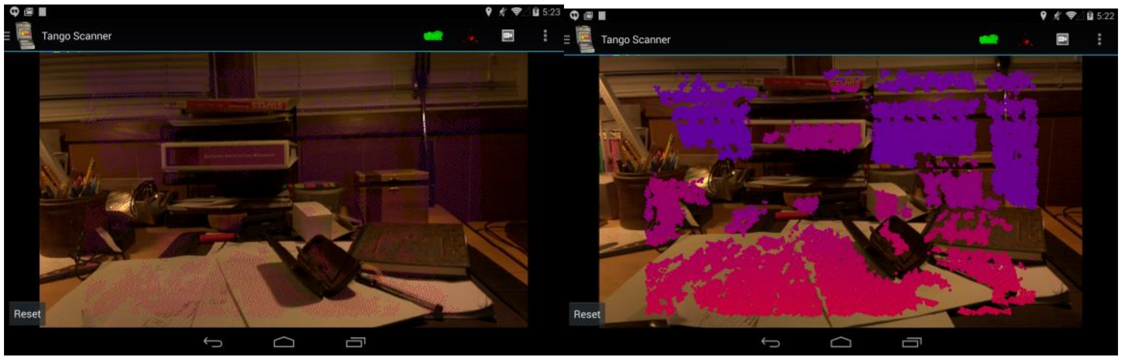
When Tango is working, you'll see the point cloud continuously updating on top of a live video image. When things aren't working so well, the point cloud will stop or disappear, or the image will, or the app will just plain crash. This is bleeding edge, after all. In general, no data collected without having the constantly updated point cloud displayed should be trusted.

The purpose of the display is to give you some reasonable assurance that you're collecting decent data. The cloud pixels are color coded by depth, and the system is more or less able to map the pixels onto their correct location on the background video image. Seeing the cloud points disappear, seeing shiny silver diagonal crosshatch patterns on the display, losing video, etc., are all ways that Tango likes to not do what you want it to do. This screen is meant to give rapid positive visual feedback on the quality of the data being collected.

You can adjust both the size of the point cloud pixels and the distance used in color coding the image. In the former case, point sizes can be reduced when the environment provides rich feedback to the Tango sensors and increased when the environment is poorly visible to Tango. In the latter case, depth color coding can be adjusted for the space you are working in, i.e. close up and intimate (for Tango) or wide open.

4.5.1 Adjusting the point cloud pixel size

The two images below show the smallest and largest values the point cloud pixels can assume. To adjust this, drag down with one finger on the **right** side of the screen to increase the size of the pixels and drag up to decrease their size. Note that dragging is relative, i.e. the system is paying attention to how far you move, not where you start or stop (beyond dragging up or down, that is).



4.5.2 Adjusting the depth colorization range.

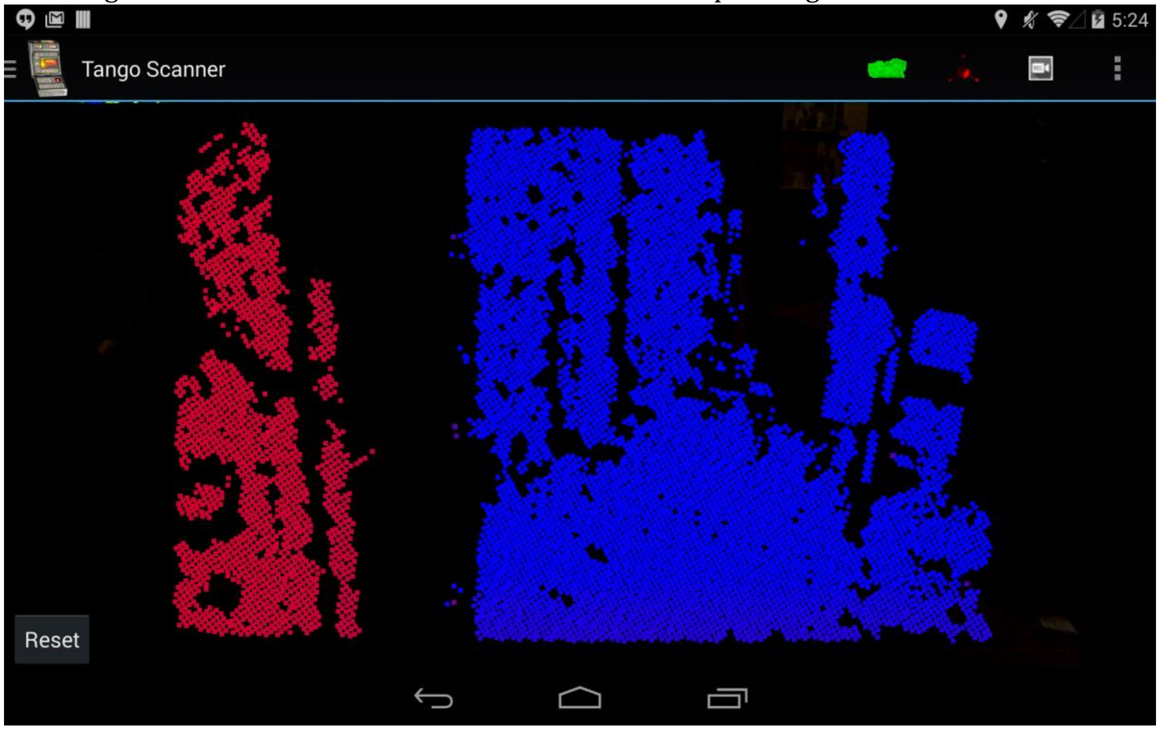
The display color codes pixels by their distance, where the furthest pixels are bright blue, the closest are bright red, and the middle is purple as shown in the gradient to the right. The top represents the closest objects, and the bottom the furthest. In order to do this, you have to have an idea about what the furthest distance **is** under certain constraints due to performance concerns. And that's where you come in, dear user. You're best suited to estimate and control that value, and you do it by sliding your finger from right to left and back again at the bottom of the screen. When you move from left to right you **increase** the depth range, and when you move from right to left you **decrease** the depth range.

The system supports a depth range from 1 to 20 meters[2]. Colorization always starts with the closest pixels being red, and then transitioning to blue over the depth range. Anything that is further than the depth range will also be blue, on the assumption it is unhappy and feeling left out.

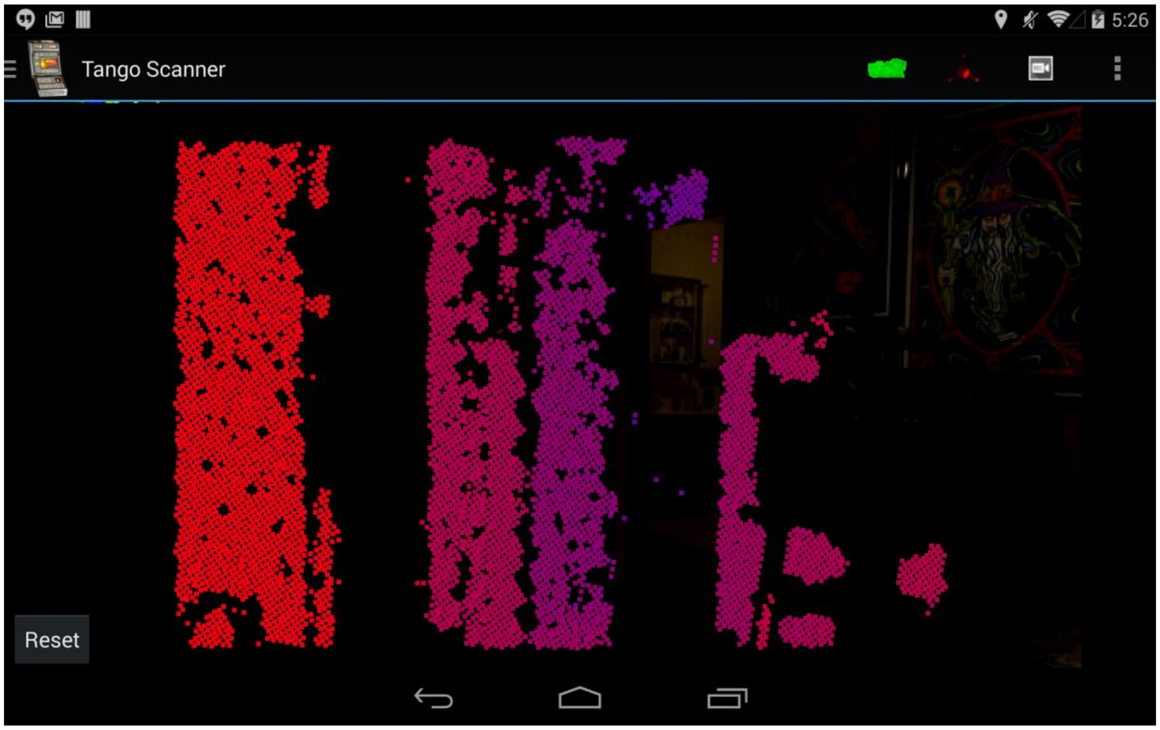
I could go on this way and explain colorization purely in terms of the math, however a few pictures help avoid equations. The picture ***wherever*** is an example where the range is too short. There is a red element in the immediate foreground, and then everything else is blue. If this represents the average volume you're going to be working in you want to drag from

[2] Every time I bump into Tangos use of measure as double precision values in meters, I am just so happy. It makes the model environments such a pleasure to work with.

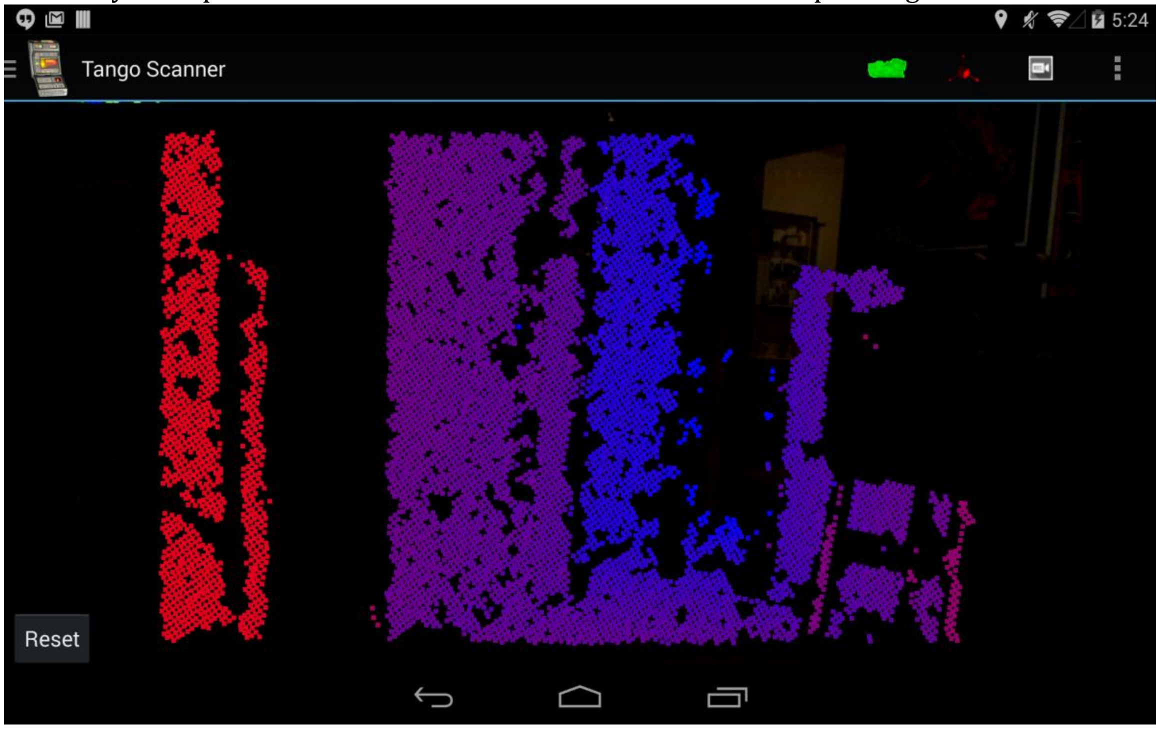
left to right at the bottom of the screen to increase the depth range.



Now, due to the sometimes sluggish behavior of the drag controls combined with our frantic pawing at the display, we shoot right past the sweet spot and end up with too large of a depth range. This is indicated when most of the foreground is red or red-ish, and only very far points have any blue to them.



Reachieving a level of calm and carefully dragging right to left at the bottom of the screen, we finally end up with a better distribution of color across the depth range.



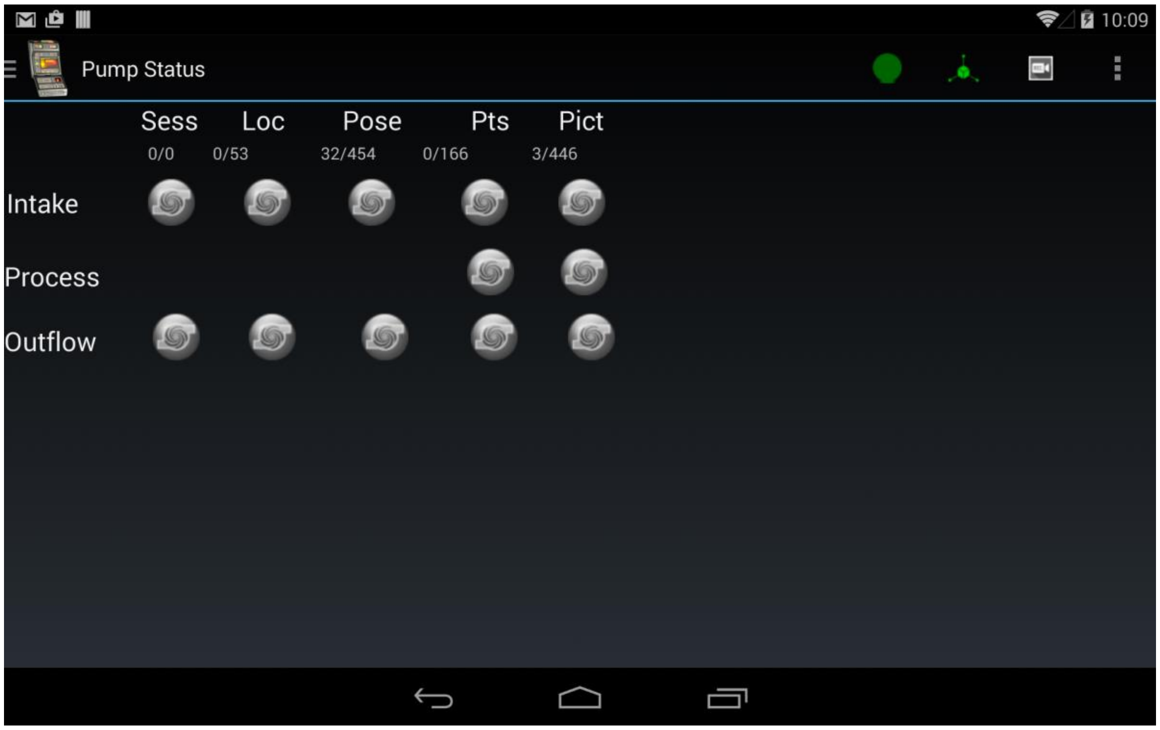
Depth ranges need to be adjusted based on the average instantaneous dimensions of your recording area. This means for a house, something on the order of the size of 2 rooms, i.e. what you can see on average at any given time.

A well calibrated depth range is of great use while recording. Alignment of the depth cued point cloud pixels with the video feed is good enough that environmental drift and corrections can be easily perceived because the cloud fails to align with the image. Now, this all said, the best range can change dynamically over a recording session, at which point you've got two choices. Dance around stabbing at the screen and trying not to excessively contribute to pose drift, or tough it out and do nothing. Either choice has its ups and downs.

4.5.3 Virtual Fish Biscuits

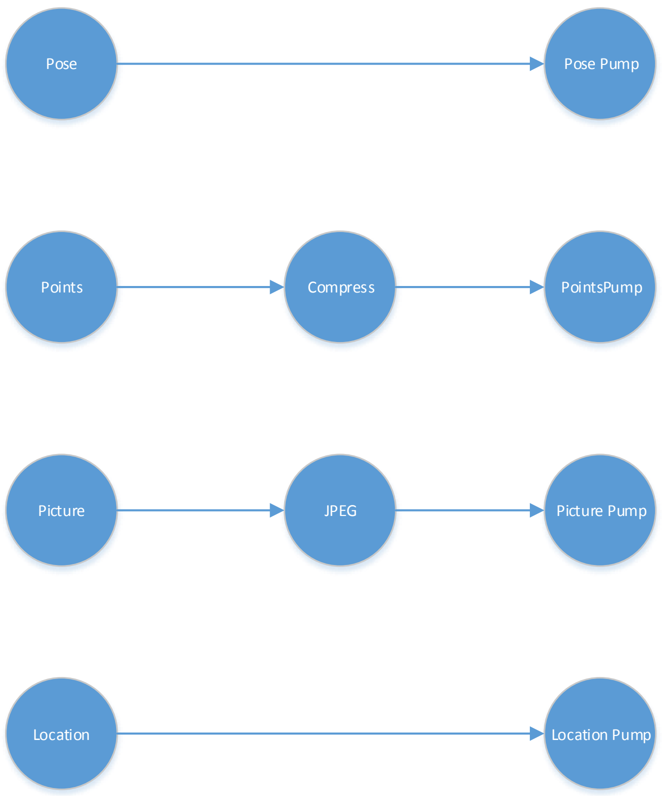
The drag mechanism does not work perfectly, and isn't likely to soon. Therefore, at the end of a drag when you have actually succeeded in changing a value, a brief toast will appear that tells you the current (changed) values for Depth Range and Point Size. This is meant as a reward for successfully accomplishing this. Without the reward, the activity is too uncertain to be of much value.

4.6 Pumps



This shows the status of the data pumping systems and various information about their efficiency. It's still under development, therefore right now it only reports key performance indicators on the intake pumps. Those are the fractional values at the top, where the numerator counts the number of dropped data packets and the denominator the total number of packets processed.

All activities with respect to pumping, especially fiddling with it, are attempts to deal with a fundamental cruelty of the system. All incoming data arrives at information buffers in the form of queues. If a piece of information shows up wanting a spot and there is no room for



it, then the oldest piece of information in the queue **is thrown away** to make room for it. Data discards arise from a number of drivers, and are deemed acceptable in many cases because the discard rate is low and the goal is large amounts of data for feeding to sophisticated reduction systems. In cases where collection time is limited or capture rates need improvement you will see a reduction in dropped data when recording to the local device. That said, the less data thrown away the better.

There are 4 independent data flows to configure, where two are single stage flows and 2 are double stage flows. The key principle is that each line represents a buffering connection, i.e. a certain number of poses, or pictures or whatnot can be held in the queue because Tango got a call or the cloud server has become sluggish.

Pose and Location pumps can be dispensed with easily. Neither data quanta is very large, as a pose is just information about location and attitude, and location is basically GPS coordinates. The primary function of the single queue between the source and the pump is to allow for things to back up a bit if the source has temporarily gone missing. If your comms are bad, or your server is too busy participating in a botnet, then certainly increase these queues. They can be made quite large with relatively low cost, as long as there's actually a point to it.

Points and Pictures are a different matter, as they represent the largest load on the system. These are basically pipelines where the fundamental quanta coming in is a blue whale. Either a raw 1Mb bitmap, or a nice fat cloud of 3 dimensional points. Hence, an internal stage is needed to compress the data so as not to disastrously overrun either comms or memory capabilities. Note that carefully. When you increase the size of the picture queue by one, you are effectively increasing its potential size by one **megabyte.** If you simply give points and pictures nice fat intake queues, and you've got a dog slow cloud service back end? All your memory is gone **and** you're back to throwing away stuff.

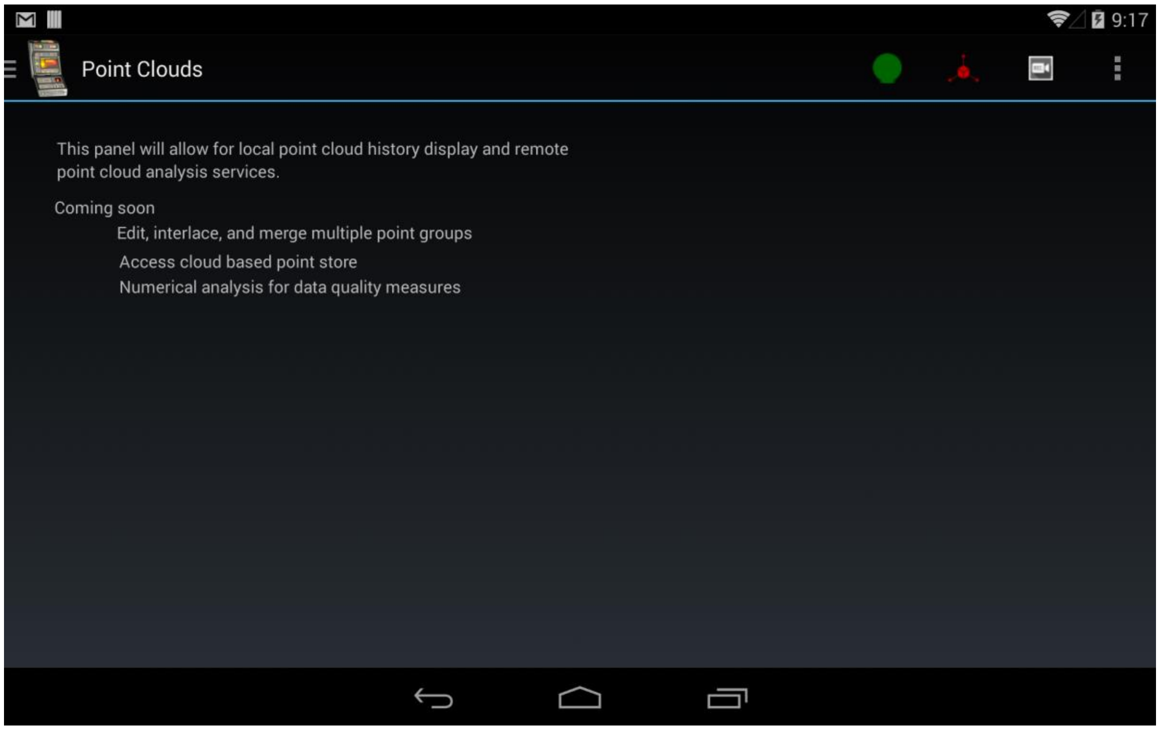
The application is built with a large heap model so there is room to maneuver. The best I can do at this time is give some practical rules of thumb.

If you don't want drops, record to memory and upload. This is the fastest pipeline by far, and the uploader is patient and doesn't throw things away. It does make it a two-step process to hoist data into the cloud, but one of relatively little trouble.

Statistically, blaming the internet works out in the majority of cases. While the act of JPEG encoding an image is certainly work, phoning home with the data is work undertaken in faith, and that doesn't always turn out so well. It's the **internal** point and picture queues that benefit most from size increases (and happily cost much less than their intake side). This is where processed data can be parked until the wireless connection begins to resemble an actual network again.

Don't sweat the drops so much - This isn't the Dyson 'I can suck the paint off a car' approach here, this is the Roomba, 'yeah, but after 30 days with me around you've been beat by a flyweight with a lot of enthusiasm'. No individual frame of data matters that much because the only purpose of an individual frame is to add to the larger whole.

4.7 Points



This panel will allow for local point cloud history display and remote point cloud analysis services.

Coming soon

Edit, interlace, and merge multiple point groups

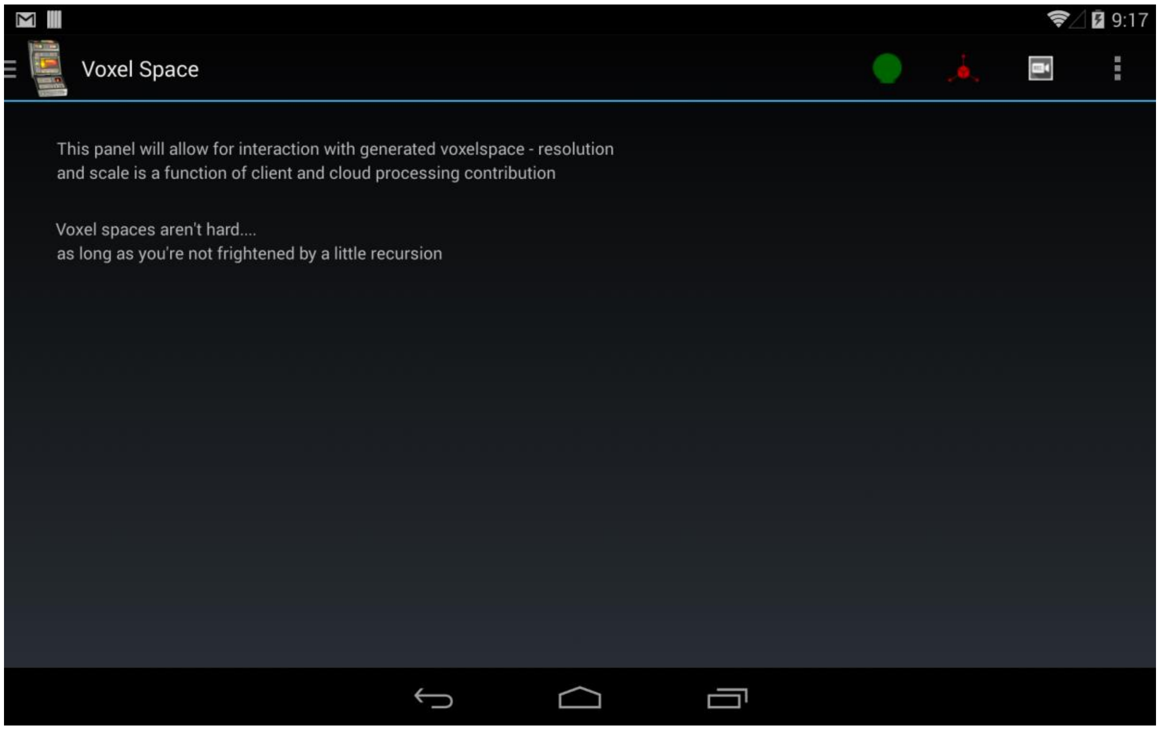
Access cloud based point store

Numerical analysis for data quality measures

In development – will become part of baseline Tango Tricorder

There are features in the offing now - if you have locally recorded data, you'll be able to bring it into this view. Yes, more than one Pose/Pointcloud. Yes it will show the data as a correctly assembled whole, to the best of its ability. No there will not be complex analytics.

4.8 Voxel Space

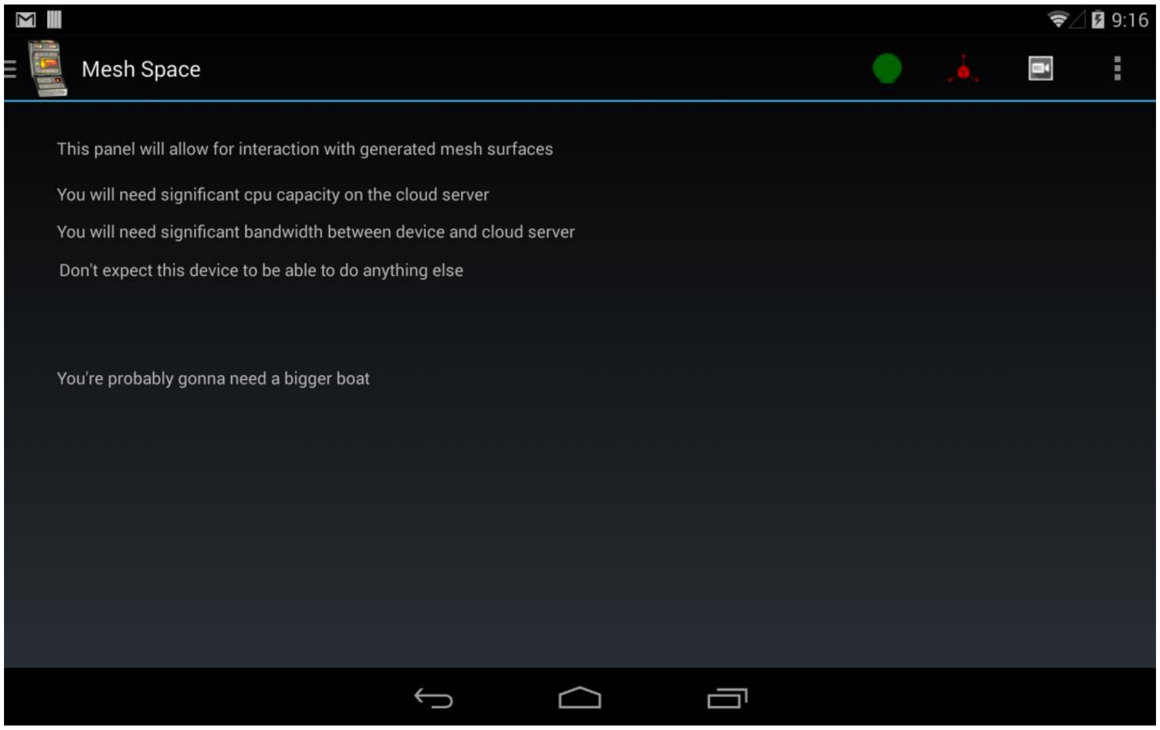


This panel will allow for interaction with generated voxelspace - resolution
and scale is a function of client and cloud processing contribution

Voxel spaces aren't hard....
as long as you're not frightened by a little recursion

In development – consumption functionality will become part of baseline.

4.9 Mesh Interaction



This panel will allow for interaction with generated mesh surfaces

You will need significant cpu capacity on the cloud server

You will need significant bandwidth between device and cloud server

Don't expect this device to be able to do anything else

You're probably gonna need a bigger boat

Envisioned – consumption functionality will become part of baseline.

4.10 Data Elements

There are five streams of information coming from Tango Tricorder, where each stream carries a sequence of a specific element.

Session	Generated on the start of each recording session
Location	Generated on each GPS location notification received
Pose	Generated on every pose check.
Points	Collected on each xyzij event from Tango
Pictures	Collected on every image event from Tango

What connects all of the data elements is time. Every element is reported with two time signatures, the long system time tick count obtained from Android and the double tick value reported by various Tango subsystems. The Tango reported tick is always cached and used for elements that do not have directly related Tango ticks, i.e. Session and Location. For all others, the tick they have is the tick Tango gave them.

There is often a pleasantly high coincidence of Tango time stamps between disparate data elements, however any reasonably strong solution will of course require some form of interpolation or estimation.

4.10.1 API

Service Check	api/tango/service	The system simply does a get on this service and expects an OK response. This is the availability test.
Session	api/tango/session	Posts Session JSON nodes to this endpoint
Location	api/tango/location	Posts Location JSON nodes to this endpoint
Pose	api/tango/pose	Posts Pose JSON nodes to this endpoint
Points	api/tango/points	Posts Point cloud JSON nodes to this endpoint

Picture api/tango/picture Posts Picture JSON nodes to this endpoint

Given you can make the relevant REST end points available, you can easily examine this data. Start up the app, turn on recording, and point it at your end points, and Bob's your uncle.

4.10.2 Common Fields in all Objects

There is a base class that contributes 3 fields that appear in every specialized structure. The definition here should be seen as what is at the head of each data item, and the logic here is effectively the **super()** call in later example code.

```
Date At;                                // Device UTC timestamp
double timestamp;                       // Tango internal timestamp
String device;                  // 64 bit ID in hex form (16 chars)
        public TangoObject(double stamp) {
                At = new Date();
                TangoObject.LatestTimestamp = timestamp = stamp;
                device = Quaestor.Singleton().mDeviceId;
        }
The Java code for reading this standard header information is;
```

```
        public TangoObject(DataInputStream s) throws IOException {

                At = new Date(s.readLong());

                timestamp = s.readDouble();

                device = s.readUTF();

        }
```

4.10.3 Session

Generated each time you start recording information. This helps frame the other collected data as well as define constants across the recording session. Most, in fact is constant with respect to the lifetime of the device. The IMU and camera information are used in reconstructing the acquisition frame of reference, and the rest of the values are primarily used in matching captured images to the 3D model.

```
{
        “deviceid” : {android device id},
        “at” : {android ticks},
        “timestamp” : {tango timestamp},

        “devPos” : [ {x,y,z from initial IMU device position}],
        “devAtt” : [ {q0,q1,q2,q3} from initial IMU device attitude}],
        “camPos” : [ {x,y,z from initial camera device position}],
        “camAtt” : [ {q0,q1,q2,q3} from initial camera device attitude}],
        “imageSize” : [ {w,h}  defining raw camera image size}],
        “focalDimensions” : [ {x,y}  defines camera focal dimensions (double)}],
        "principalPoint" : [ {x,y}  defines pinhole location}],
        "distortion" : [ {c0,c1,c2,c3,c4}  lens distortion coefficients}]
}
```

4.10.3.1 Session Raw Format

The following Java functions show how raw session data is read and written.

```
        public TangoSession(DataInputStream s) throws IOException {
                super(s);
                int imuVals = s.readInt();
                imuData = new double[imuVals];
                for(int i = 0;i < imuVals;i++)
                        imuData[i] = s.readDouble();
                int camVals = s.readInt();
                camData = new double[camVals];
                for(int i = 0;i < camVals;i++)
                        camData[i] = s.readDouble();

        }
        public void publishToFile(DataOutputStream  s) throws IOException {
                super.publishToFile(s);
          double[] imuData = TangoJNINative.getIMUFrame();
          s.writeInt(imuData.length);
          for(int i = 0;i < imuData.length;i++)
                s.writeDouble(imuData[i]);

          double[] camData = TangoJNINative.getCameraIntrinsics();
          s.writeInt(camData.length);
          for(int i = 0;i < camData.length;i++)
                s.writeDouble(camData[i]);
        }
```

4.10.4 Location

This is location information received from the Android location service.

```
{
        "deviceid" : {android device id},
        "at" : {android ticks},
```

“timestamp” : {tango timestamp},

“latitude” : {android latitude},

“longitude” : {android longitude},

“elevation” : {android elevation},

“provider” : {android provider}

}

Location Raw Format

The following Java functions show how raw session data is read and written.

```
public TangoLocation(DataInputStream s) throws IOException {
        super(s);
        latitude = s.readDouble();
        longitude = s.readDouble();
        elevation = s.readDouble();
        provider = s.readUTF();
}

public void publishToFile(DataOutputStream s) throws IOException {
        super.publishToFile(s);
        s.writeDouble(latitude);
        s.writeDouble(longitude);
        s.writeDouble(elevation);
        s.writeUTF(provider);
}
```

4.10.5 Pose

{

This is pose information acquired from the Tango services.

Note that Quaternion information from Tango is not swizzled, ie. **s**, or more commonly w, trails the xyz coordinates.

“deviceid” : {android device id},

“at” : {android ticks},

“timestamp” : {tango timestamp},

“Quality”: {non-zero if the pose is localized},

```
        "Px": {pose x},
        "Py": {pose y},
        "Pz": {pose z},
        "AQ0" : {pose quat q0 },
        "AQ1" : {pose quat q1 },
        "AQ2" : {pose quat q2 },
        "AQ3" : {pose quat q3 },
        "FrameOfReference" : {pose adf guid id }
}
```

4.10.5.1 Pose Raw Format

The following Java functions show how raw pose data is read and written.

```
        public TangoPoseData(DataInputStream s) throws IOException {
                super(s);
                quality = s.readByte();
                posX = s.readDouble();
                posY = s.readDouble();
                posZ = s.readDouble();
                attX = s.readDouble();
                attY = s.readDouble();
                attZ = s.readDouble();
                attW = s.readDouble();
                poseContext = s.readUTF();
        }

        public void publishToFile(DataOutputStream s) throws IOException {
                super.publishToFile(s);
                s.writeByte(quality);
                s.writeDouble(posX);
                s.writeDouble(posY);
                s.writeDouble(posZ);
                s.writeDouble(attX);
                s.writeDouble(attY);
                s.writeDouble(attZ);
                s.writeDouble(attW);
                s.writeUTF(poseContext);
                s.flush();
        }
```

4.10.6 Points

The raw float array is compressed (deflated) via zlib, and then time is wasted by base64 encoding that. However, the end result is still vastly better than raw uncompressed text.

The ***endian-ness*** of the receiving machine will matter. For example, WinTel requires that I reverse the order of each floats bytes, but not the floats themselves. In other words, for each 4 bytes that comprise a float, those four bytes need to be reversed if you're receiving them on a WinTel box.

The numfloats value counts the number of **floats,** i.e. it is 3x the number of points in the cloud.

```
{
        "deviceid" : {android device id},
        "at" : {android ticks},
        "timestamp" : {tango timestamp},
        "NumFloats" : {number of scalar values in the payload},
        "CompressedPoints" : {deflated RAW float array then base64 encoded}
}
```

4.10.6.1 Point Cloud Raw Format

The following Java functions show how raw point cloud data is read and written.

```
        public TangoPointCloud(DataInputStream s) throws IOException {
                super(s);
                numFloats  = s.readInt();
                int datalen = s.readInt();
                compressedData = new byte[datalen];
                s.read(compressedData);
        }
        public void publishToFile(DataOutputStream s) throws IOException {
                super.publishToFile(s);
                s.writeInt(numFloats);
                s.writeInt(compressedData.length);
                s.write(compressedData);
                s.flush();
        }
```

4.10.7 Picture

```
{
        "deviceid" : {android device id},
        "at" : {android ticks},
        "timestamp" : {tango timestamp},
        "Width" : {width of the picture},
        "Height" : {height of the picture},
        "Depth" : {byte depth of the picture, invariant, 4},
        "Image" : {jpeg representation, then base64 encoded[3]},
}
```

4.10.7.1 Picture Raw Format

The following Java functions show how raw image data is read and written.

```
                public TangoPicture(DataInputStream s) throws IOException {
                        super(s);
                        width = s.readInt();
                        height = s.readInt();
                        int nbytes = s.readInt();
                        rawJPG = new byte[nbytes];
                        s.read(rawJPG);
                }
                public void publishToFile(DataOutputStream s) throws IOException {
```

[3] Yes, I realize that this is gruesome at many levels – that said, it's easy, and it's inspectable.

```
                super.publishToFile(s);
                s.writeInt(width);
                s.writeInt(height);
                s.writeInt(rawJPG.length);
                s.write(rawJPG);
        }
```

5 Issues

Session pump counts are currently meaningless.

Crashes

Version 1 on an experimental platform. Whodathunkit ? That said, I know of an issue in openGL that gets the system wound up, and there are certainly issues that arise if Tango wants to be a bit fussy.

Data Volume

Yup. Yup. The best current limit on data volume is battery lifetime. If you are constantly recording, your battery is getting hammered. I do want to cut down the comms volume by moving to pure binary delivery of point cloud and image data, however that involves splitting up the JSON, which complicates the streams, so that makes premature optimization the root of all evil.

Boot screen stays up - yes, but at least you get a toast from the defeated system begging you to drag open the navigation drawer it.