**Graduate Projects**

University of Colorado at Boulder

Aerospace Engineering Sciences

ASEN 5018/6028 –Fall 2015

|  |
| --- |
| **FlyNet**  **Perception Subsystem Summary/Continuity Document** |

|  |  |  |
| --- | --- | --- |
| **Approvers List** | | |
|  | Title | Name |
| Prepared By | Perception Team |  |
|  |  |  |
| Approved By | Perception Lead |  |
| Approved By | Systems Engineer |  |
| Approved By | Project Manager |  |

**1: Introduction & Summary**

The goal of the perception subsystem is to provide the onboard infrastructure needed to implement target identification and provide online updates to the a priori map.

We have implemented a simultaneous localization and mapping (SLAM) solution using a hardware-assisted stereo vision setup. The SLAM solution, known as RealTime Appearance-Based Mapping (RTABMap), provides the onboard navigation algorithms an estimate of the UAV's currrent position in the world map. Target identification is accomplished via a FLIR Lepton sensor, while the hardware stereo setup is provided by a DJI Guidance subsystem.

# **2: Semester Report**

## 2.1: Objectives and Tasks List

Here is where you will list **ALL** goals and tasks that you’ve either been assigned or have determined yourselves, **complete or not**. Tie to a requirement where applicable.

**Completed**:

1. Completed a trade study of perception sensors and associated SLAM techniques
2. Demonstrated a functional SLAM solution appropriate to flight hardware limitations without UAV integration
3. Implement sufficient glue ROS nodes to enable system integration between Guidance hardware and Pixhawk autopilot
4. Demonstrated functional SLAM solution onboard flight hardware
5. Obtain images from FLiR camera
6. Implement tracking code to identify and track target positions
7. Interface FLiR with Odroid
8. Create image publisher and subscriber nodes for FLiR images

**Incomplete**:

1. Closing the loop between flight control and planning using SLAM outputs
2. Updating a priori map with SLAM results
3. Update tracking code to use tf and subscribe to ROS image nodes
4. Use Odroid to obtain images from FLiR sensor

## 2.2: Issues

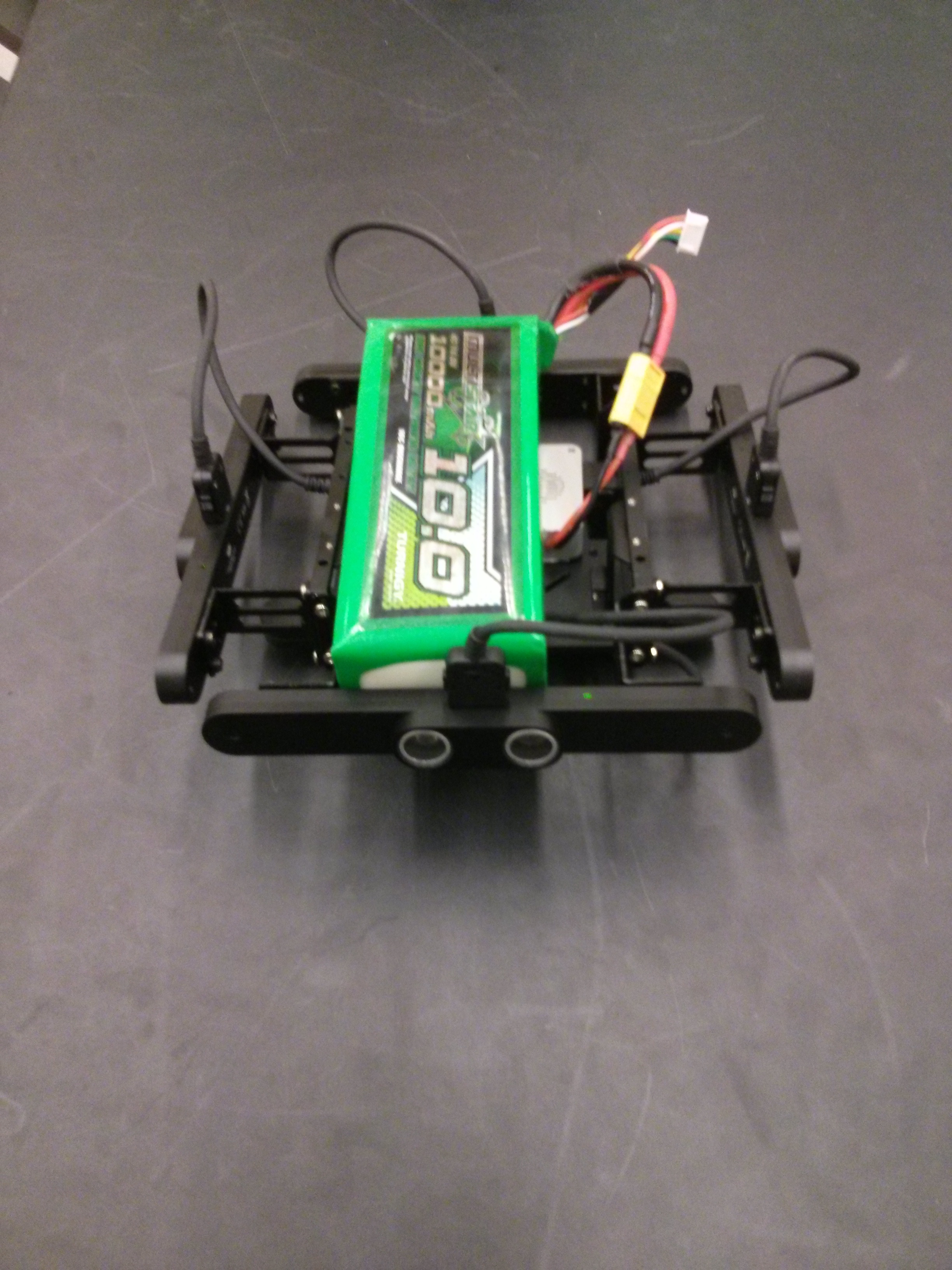
1. Many implementation problems stem from the use of the DJI Guidance sensor. While this sensor has a strong potential, its limited market penetration has led to underdeveloped software support that had to be backfilled by team members. While alternative sensors exist that could readily be used to implement SLAM, Guidance is able to offload many of the most CPU-intensive tasks and preserve the onboard execution budget. Guidance was unavailable until approximately midway through the semester, shortening the amount of time available to identify and solve integration problems.
2. Planning and map update integration tasks remain incomplete due to time constraints.

## 2.3: Lessons Learned

1. All block-matching stereo vision algorithms include a blank area on the left of the depth image as a side effect.
2. Passive stereo vision is fundamentally more constrained than structured light active sensing due to a requirement for feature correspondences.
3. Having a technology in which only one person on the team is familiar presents management difficulties to prevent burnout and/or task stovepiping.
4. Binary-only blobs are a recipe for disaster.
5. More time is often spent troubleshooting communication protocols than developing the algorithms and code. Start on this as early as possible.
6. Should have ordered Odroid earlier in the semester so development could begin earlier.

## 2.4: Procedures

The perception subsystem is implemented in three layers: mechanical, logical, and electrical. SLAM is implemented using Guidance's outputs, while targeting uses the FLIR output coupled with the current pose reported by SLAM to localize targets. The remainder of this section assumes the user is familiar with ROS, a data distribution and processing middleware layer; the reader is directed to [www.ros.org](http://www.ros.org/) for further information and background. All nodes are implemented in ROS Indigo.

Figure 1: Guidance Sensor

Mechanically, the Guidance system has its own carrier that support five sensor nodes and its central processing core. Four nodes face in cardinal directions, while the fifth faces downward. Each node consists of a stereo pair as well as an ultrasound sensor. Further information is available from DJI at <https://developer.dji.com/guidance/>

Electrically, Guidance is capable of directly being driven by the onboard 4S LiPo battery; Figure Figure shows a testing configuration with a battery directly connected to Guidance via an XT60 connector. Data are output from Guidance using either serial connection or a USB connection. Due to bandwidth limitations of the serial interface, we used the USB connection via a micro-B to A cable to the onboard CPU.

Logically, the USB connection does not implement any standard device classes; rather, the vendor defines a set of custom URBs. Of note is that DJI does not have an officially sanctioned VID from the USB Implementer's Forum, which could lead to VID/PID conflicts in the future. Each of the custom URBs is accessed via a libusb-1.0 based API that is only distributed in binary form from DJI. Currently, x64, x32, and ARMv7 are available on Linux. Via the API, Guidance's sensor information is available, including ultrasound, rectified imagery, visual odometry, and IMU data.

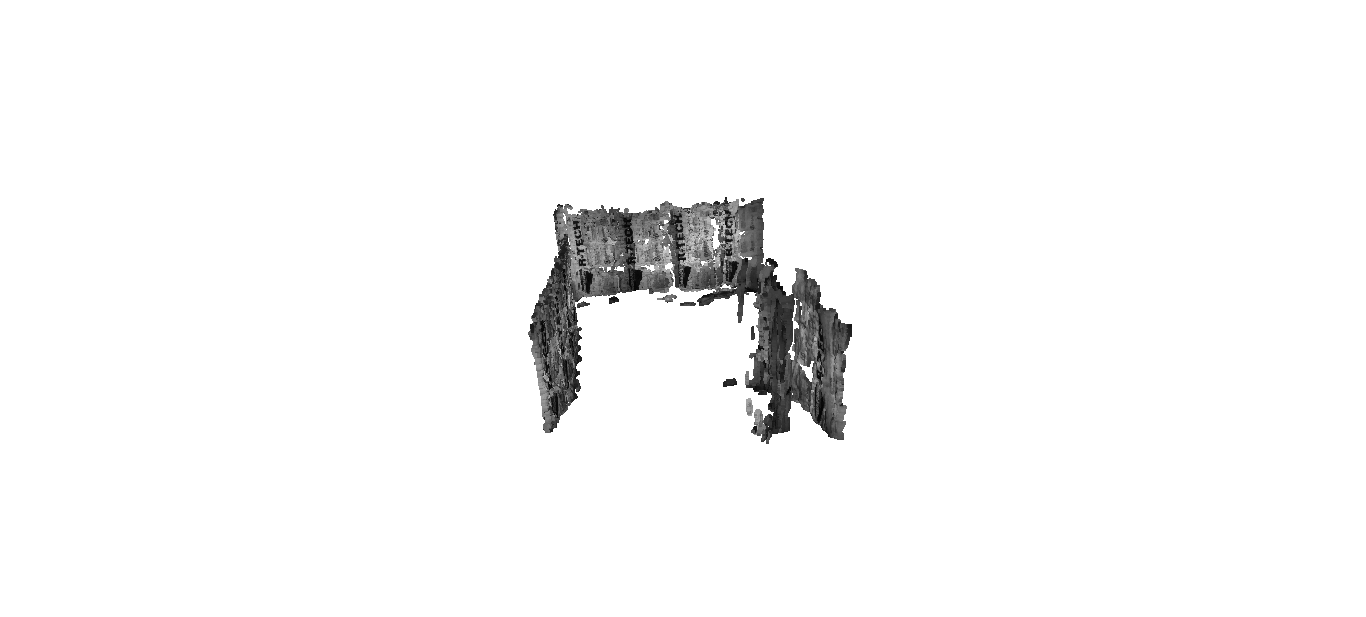
In our testing of the Guidance system, the onbaord block-matching based stereo pipeline produces inferior results that are unusable for the purposes of depth-based SLAM. The depth images are sparse and riddled with inconsistencies that lead to extensive mapping errors. As a result, we implemented a workaround that utilizes Guidance to produce visual odometry, but instead uses the rectified stereo pair as input to RTABMap. This pipeline shows promise and is a realistic solution going forward.

As provided by DJI, the Guidance ROS node was unusable and required several modifications. The updated source is available through the BitBucket repository under <https://bitbucket.org/cuflynet/guidance-sdk-ros.git>, while the upstream is <https://github.com/dji-sdk/Guidance-SDK-ROS.git>. Our customizations include the addition of code to cleanly shutdown the Guidance interface and prevent a situation where the Guidance core does not respond to new connection requests, as well as adding runtime configuration flexibilty to change operations without requiring code recompilation.

The bulk of the SLAM operations are provided by RTABMap running onboard the host CPU, available at <http://introlab.github.io/rtabmap/>. ROS nodes are provided that nicely wrap the underlying codebase and provide a clean interface to sensor inputs and SLAM outputs while providing for headless operation (so that no UI need be rendered aboard the UAV). The primary configuration required of RTABMap was to ensure that all needed ROS topics were publishing, either from the *guidance* ROS node directly or from intermediate processing nodes.

As a first cut, RTABMap was run against an Asus Xtion Live Pro RGBD camera that utilizes structured light to produce a depth image. RTABMap was able to produce a reasonable map of the Fleming workspace using this input while running aboard a laptop, but demonstrated high CPU utlization related to the computation of visual odometry.

Once the Guidance system was available and its ROS node was usable, RTABMap was run against a single image and the corresponding depth image. The produced map was useless, a mass of blobs. The mitigation strategy was to migrate processing to the host in an incremental basis to attempt to recover function and minimize impact on host CPU.

Figure 2: RTABMap Output of a basic hallway

The next strategy was to use a rectified stereo pair produced by Guidance as input to RTABMap, generally following the tutorial at <http://wiki.ros.org/rtabmap_ros/Tutorials/StereoOutdoorMapping> and adjusting parameters as necessary to conform to our system architecture. An interim result is shown in Figure Figure that clearly shows a nominal hallway and a clear space between walls. This result was produced using a laptop instead of flight hardware; the CPU utilization of a flight hardware solution has yet to be characterized.

It was clear from the initial project description that some sort of target tracking and identification (TTI) would be required to complete this project. When the targets were narrowed down to be human, it became obvious that a thermal tracking system would have a major advantage given the thermal signature of a human. After much research and trade studies, the FLiR Lepton camera was chosen due to its minute size, weight, cost, and power requirements. It is also better documented for development by amateurs than other options.

The TTI subsystem of perception is based largely around the FLiR sensor. This sensor is a 60x80 pixel long wave infrared camera mounted to a breakout board. The FLiR sensor can be configured over the I2C pins on the breakout board, while the images themselves are sent over SPI. Unfortunately, Automatic Gain Control (AGC) is enabled by default and may present problems to the tracking algorithms discussed in the following paragraphs. Fortunately AGC can be turned off, but this has not yet been tested and is still a wildcard.

Development of the TTI algorithms began after the sensor type was determined. Two main methods of tracking were considered. A discrimination and time-history based method using Kalman filters was determined to be overkill for this project since only the current positions of targets are required.

An Arduino Teensy 3.0 was used in order to become familiarized with the FLiR camera and obtain the first few images from it. This was an arduous process but it has provided a basis of understanding of the communication protocols and performance. In order to speed up the development and testing of the TTI algorithms, a MatLab program was developed to generate simulated FLiR images and Guidance depth maps of a three-dimensional environment. The purposes of the images will become clear in the following paragraphs. Real life testing of the FLiR camera have shown that it is fully capable of detecting a human sized object at the required 5 meter distance.

Figure 3: Sample FLiR image

The FLiR camera will be controlled by a C++ program on the Odroid. This program controls initial configuration, communication over SPI and I2C, and publishes the images to a ROS node. The target tracking and identification (TTI) C++ program subscribes to this node to obtain the images. TTI utilizes OpenCV for much of its image processing tasks. The task of communicating with the Odroid not yet been completed, but code has been written and must now be tested.

TTI identifies and tracks an object through the following steps. Using a known thermal masking range, the FLiR images are masked to only show objects that are close to human body temperature. The contours of the masked image are found and are determined to be potential targets. The targets are narrowed down and reported after filtering based on size and distance to the object. A three-dimensional position is found by utilizing the depth maps from Guidance, obtained by subscribing to a ROS node. This information is then transformed into global coordinates and published to a ROS node for use by the navigation subsystem. All of the TTI algorithms have been written and tested, but it must now be integrated with ROS.

Table 2: Software list

|  |  |  |
| --- | --- | --- |
| **Program Name** | **Version** | **Purpose** |
| **Eclipse IDE** | 4.5.0 | Code development |
| **MatLab** | 2R2013a | Image simulation |
| **Arduino IDE** | 1.6.6 | FLiR testing through Teensy 3.2 |
| **Teensyduino** |  | Teensy 3.2 functionality with Arduino IDE |
| **RTABMap** | 0.10.10 | SLAM Solution |
| **GNU Emacs** | 24.5.1 | Code development |
| **ROS** | Indigo | Data distribution and processing middleware |

# **3: Next Semester/Future Expectations**

## 3.1: Prioritized List of Tasks and Objectives

1. Close the loop around SLAM
2. Communicate with FLiR through Odroid
3. Finalize ROS nodes for images and position publishing and subscribing
4. Create startup script for TTI subsystem
5. Add smarter filters for target identification to reduce false targets.
6. Integrate target tracking with SLAM solution
7. Start researching multiple map integration

## 3.2: Starting Points

1. Right now, enough glue code has been written to allow Vicon to provide a position estimate to the Pixhawk while monitoring the SLAM position solution. The next step is to characterize the accuracy of the SLAM solution by logging both concurrently and examining the SLAM solution for drift.
2. Target tracking has been developed under the assumption that depth information is coaxial with the FLIR sensor. This is a crude approximation that can be resolved via appropriate calibration, but requires a calibration target visible in both far IR and visual. Once obtained, the mounting transformation can be used to produce world coordinates of identified targets.
3. Multiple-master ROS has been explored through the <http://wiki.ros.org/rocon_multimaster> project – this would seem to be a reasonable place to start development.
4. Take a look at the code located at <https://github.com/groupgets/LeptonModule/tree/master/software/beagleboneblack_video> for an example of how to communicate with FLiR via a Beaglebone Black. This may be able to be cannibalized for use with the Odroid.
5. Use this tutorial to build the publisher and subscriber. It should work with very small tweeks: http://wiki.ros.org/image\_transport/Tutorials/PublishingImages.
6. Combine all necessary commands into one script to start up the TTI system.
7. Area of a contour is currently the only method used to determine valid targets. Update this to use the depth map and the area in order to distinguish between near and far targets.

## 3.3: Improvement, Updates, Verification

1. RTABMap doesn't explicitly use GPU acceleration to aid its processing; one improvement would be to write critical algorithms as GLSL ES 3.30 shaders or OpenCL 1.1 Full Profile kernels to utilize the onboard GPU in addition to the CPU.
2. TTI assumes that the FLiR image and the depth map are provided at the same instance. Put in checks to make sure that they are sufficiently close together in time that they should correspond well.
3. TTI assumes that automatic gain control (AGC) is turned off. Perform tests to determine if AGC should be turned off or on.