Massachusetts Institute of Technology
Lincoln Laboratory
and
Worcester Polytechnic Institute

Mobile Motion Capture

Matthew Costi
Robert Dabrowski

December 15, 2014

This work is sponsored by the Collaborative Robotics program at MITLL

Lexington Massachusetts

This page intentionally left blank.

ACKNOWLEDGMENTS

Thank you Nick Armstrong-Crews for mentoring with every step of the project, Professor Michalson for advising, and Mike Boulet for providing technology in the Autonomous Systems lab. David Scott for help with 3D printing and speeding up our prototyping process.

EXECUTIVE SUMMARY

As robotics and augmented reality research continue to advance, a common problem that appears is that of global localization for each components in the system. One solution for this is to use a motion capture room to track all of these components. This is done by marking objects with reflective fiducials that are very visible to infra-red (IR) cameras, which then work together to track the 6 degree of freedom (DoF) poses of these objects. An inherit problem with motion capture rooms is that the space in which you can track objects is limited to only a single room. To increase the capture volume, additional cameras could be purchased and placed more sparsely. This is not a practical solution as it takes about a dozen cameras to cover a single room and each standard camera costs over one thousand dollars; the cost to increase space adds up very quickly. Even more, it is difficult and often impractical to move or expand an existing system to other spaces due to the lengthy calibration process.

This project's goal was to create a mobile motion capture system that would use a single device as opposed to having multiple static cameras, allowing for theoretically infinite capture volume. The mobile motion capture system we developed is capable of tracking the same reflective fiducials as a normal motion capture system, but is limited to simple planar objects that must be completely in the field of view of the device; what we gained in mobility and low cost was traded off with with the ability to track more complex objects. An example of a trackable plane in our system is in Figure 1. The configuration of points which are called constellations, contain 4 lines; each line is a unique configuration of 4 points which we call sub constellations.

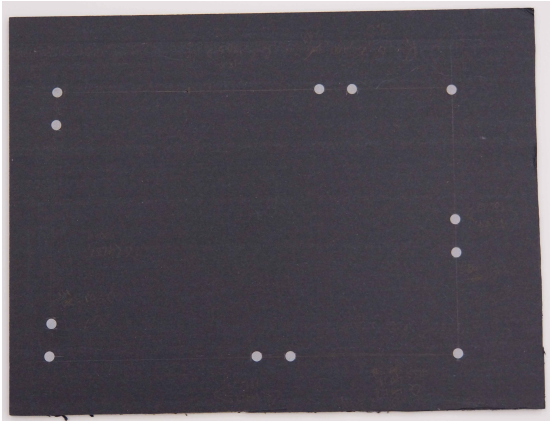


Figure 1. A planar constellation that our system tracks, which is composed of 4 unique lines

The device we are using which enabled our project is the Google Project Tango, a project currently being developed by the Advanced Technologies And Project team at Google, or ATAP for short. The Tango device uses visual odometry to estimate its own 6DoF pose relative to where the device was turned on. This grants us a global reference frame just like using a motion capture room provides. In addition to estimating its pose, the Tango has an IR camera that it uses for sensing depth. Instead, we use the same IR camera in order to sense the reflective fiducials on our

planar constellation. With the pose of the Tango and IR image data, and by calculating the pose of the planar constellation with relation to the phone, we now know where the planar constellation is relative to our global reference frame.

The system we built focuses on tracking the planar object and estimating its pose using only the IR image stream. We accomplished this by developing the system as shown in our software architecture diagram, Figure 2. The software for our system is broken down into 3 major components. The first is to search our IR images for the reflective fiducials, and get the 2D image coordinates of them. The next stage in our system takes these image coordinates and then attempts to match them to known constellation and then identifies it as such. Once it's identify is known the pose of the constellation can then be estimated.

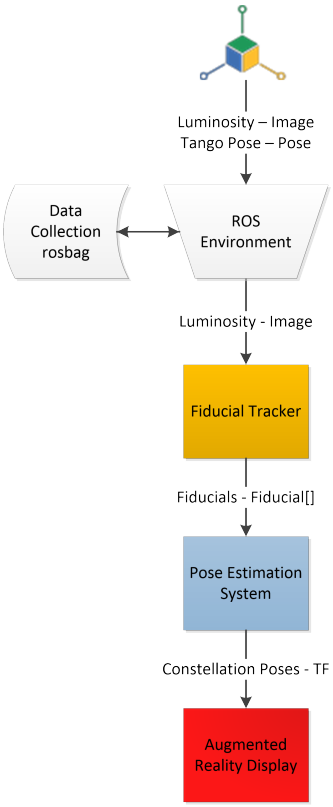


Figure 2. System overview of the major components in the Mobile Motion Capture pipeline.

In order to search the IR images for fiducials and the 2D coordinates of each, the images were first converted into binary image by applying a threshold filter where pixels brighter than the threshold are turned white, and lower are turned black. To actually search the binary image, an image processing technique known as connected components region searching is performed. The connected components search scans the entire binary image looking for connected white pixels. If pixels are connected it means the entire region of pixels is a single fiducial, so the centroids of these regions are the 2D pixel coordinate of the fiducials in the scene.

The next step in our system is to identify the specific constellation, if any, that is currently in front of the camera. A constellation is made of 4 uniquely configured lines, called sub-constellations. Each of these sub-constellations has to be identified in order to determine the identify of the entire constellation plane. Before this can be done, all of the lines in the image have to extracted. Research on line identification was conducted, leading us to implement our own version of the RANSAC algorithm that fits all fiducials to lines that lie within a defined tolerance of the line.

We chose to use lines of 4 uniquely spaced points as our sub-constellation due to the ease in distingushing them from eachother. By making use of the cross-ratio, we could measure the relative distances of each of the points in our lines, and compute the same cross-ratio value no matter how that line is projected into 2D image space. The cross-ratio is defined in eq. (1). With each sub-constellation identified, identifying the entire constellation plane involves checking if there is a constellation that exists that is made up of these sub-constellations.

$$R(A, C; B, D) = (AC/BC)/(AD/DB) \tag{1}$$

Knowing which constellation is in the image means we can proceed to estimate the pose of that constellation. Since our constellations are limited to planar surfaces, we used a method called Perspective-n-point, or PNP for short. PNP pose estimation works by knowing the local positions of points on the the plane, relative to its centroid. With this information, PNP can compare the found points to the known points, which we stored in a database, in order to give an estimate of its 6DOF pose.

Once implemented and tested, it was determined that there was about 3 cm in positional accuracy, with the object being tracked 90% of the time. The target plane could be tracked up to +/- 35 degrees in skew, and identified at any roll rotation. It could also track our object in the range of 0.5 - 3.5 meters. The entire system runs at about 10 Hz live, as this is the rate the Tango can output images; our tracking software can actually run closer to 80Hz. With these results we believe the system we developed is a great success, capable of being a great tool for tracking and localization.

To demonstrate the capabilities of this system, we developed a simple mobile robotic application that uses the systems ability to track objects. We put the the Tango device on a pan and tilt turret, which was mounted on a mobile robot. The idea was to have the mobile robot follow a tracked object. This application is split into two main components: the pan and tilt turret control, and the driving behavior.

The pan and tilt controller was in charge of ensuring that the tracked object is always in view of the Tango device. This means moving the servos to keep the object in the center of the Tango's camera frame. This was done by calculating the angular displacement between the object pose to a static reference frame in front of the the Tango.

The turtlebot driving is independent of the pan and tilt controller, and defines how the mobile robot should move in relation to the tracked object. The goal of the driving controller is to make sure the mobile robot stays at a goal position 1.5 meters in front of the object at all times. Two

basic rules define the driving behavior; drive towards the goal in an arc and rotate to face the constellation once at the goal.

This page intentionally left blank.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Page

LIST OF FIGURES

LIST OF FIGURES
(Continued)

LIST OF TABLES

LIST OF ALGORITHMS

1. INTRODUCTION

This Major Qualifying Projects (MQP) seeks to develop a solution to the localization problem in augmented reality applications by providing the ability to track moving objects relative to a global reference frame, but with the added benefit of being a completely mobile system.

The introduction to this project is broken up into multiple sections, because our project spans many different areas. Section 1.1 introduces the concept of motion tracking and Section 1.2 describes some of the basics of localization. Augmented reality, which will be supported by our mobile motion capture system, is defined in Section 1.3. Finally, two of the tools we will be using, Project Tango and Robot Operating System, are briefly described in Sections 1.4 and 1.5.

The rest of this paper is broken up into seven sections. Sections 2 and 3 provide insight into why we worked on this project and some other projects similar to ours. Section 4 states the problem we worked to solve and the goals we set for ourselves. Our methods for implementation and results are in Sections 5 and 6. Finally, we discuss our results and observations and provide insight for future work in Section 7.

1.1 MOTION TRACKING

The main interest of this Major Qualifying Project is motion tracking or motion capture (mocap). Motion capture is the ability to track and record an object's motion in three dimensional space. One popular way of doing this is by using a large number of cameras around the perimeter of a room, and having them work together to triangulate the positions of objects in the scene. Specifically, high quality infrared (IR) cameras like the one in Figure 3 are used. Figure 4 shows a retro-reflective fiducial marker which can be seen very clearly by the IR cameras since each camera has a ring of IR-LEDs that floods the room with IR light.

Figure 3. Motion Capture camera used as one of many in a typical motion capture system.

Figure 4. retro-reflective fiducial marker reflects infrared light, making them very visible to IR cameras

One widely used application for motion capture is in the film industry, namely for animation purposes. An actor would wear a suit with these reflective fiducials placed on known joints. Through motion capture the motion of the actor, which can then be used as a model for the movement of a digital character [1].

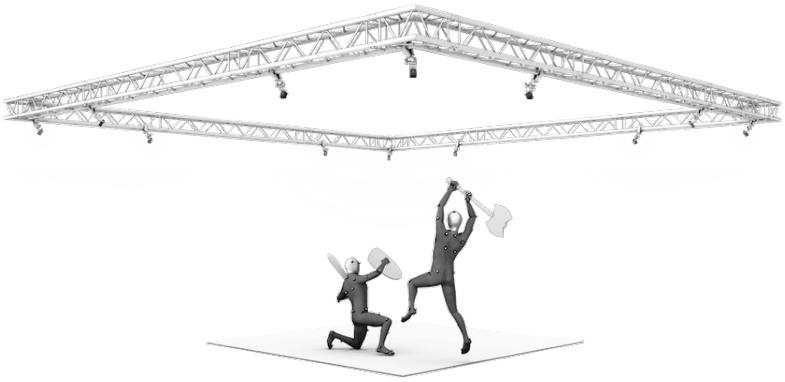


Figure 5. A typical motion capture system showing how actors motions can be tracked.

Another use for motion capture that is of more interest to this MQP is its ability to give six degree of freedom (DoF) pose estimates for pre-defined constellations, or configurations, of these markers. These constellations of markers can also be referred to as "rigid bodies". Knowing the 6 DoF pose of any rigid object of interest is very important in the field of robotics and Augmented reality as it solves the localization problem which is discussed in Section 1.3.

1.2 TECHNIQUES FOR SOLVING REFERENCE FRAMES

The end goal of localization is to calculate a homogeneous transformation matrix that specifies the rotation and translation of each object from some global reference frame. Equation (2) shows the transformation from a reference frame to an object as a matrix containing R and T, where R is the $3x3$ rotation matrix and T is the $3x1$ position or translation vector of the object from the reference frame.

$$T_{ref}^{obj} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \tag{2}$$

Figure 6 shows a tree representation of known transformations. Each node is an object or the global reference frame. Each arrow is a known transformation, which could either be calculated by a motion capture lab, a robot's odometry, or calculated in advance with known measurements to a stationary object. Here "global" is the global reference frame that all objects measure their global positions with respect to.

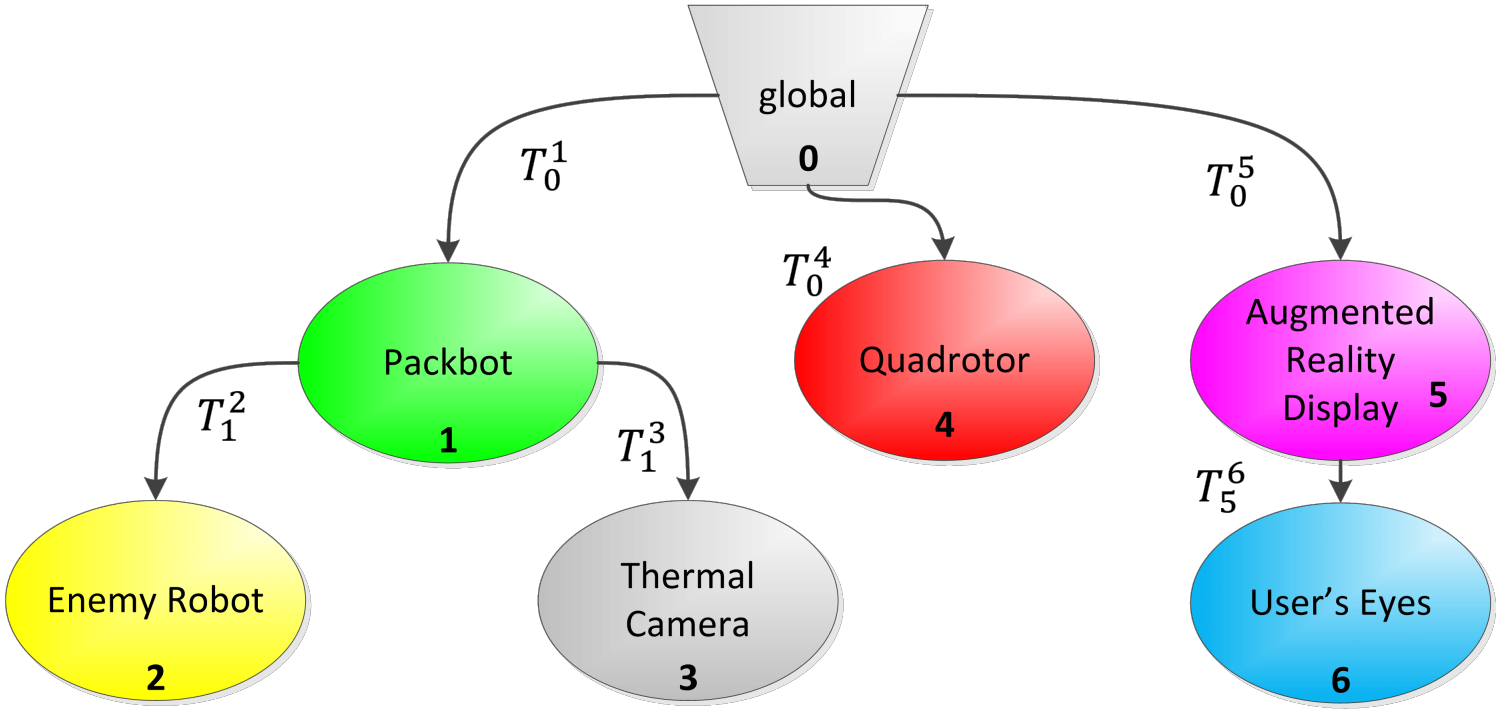


Figure 6. Transformation tree used to show the how known positions of objects relate to each other in a global frame.

Equation (3) illustrates the calculation of T_5^1, which is the position of the Packbot with reference to the augmented reality display. This is useful when trying to display or highlight the Packbot on the augmented reality display. The final transformation can be calculated by multiplying intermediate transformations in the correct order, starting with the desired reference frame and ending with the object of interest. Note that the inverse is taken of the first transformation because its original direction was in the opposite direction that is desired, shown by the arrow in Figure 6 pointing from "global" to "Augmented Reality Display".

$$T_5^2 = T_0^{5^{-1}} * T_0^1 * T_1^2 \tag{3}$$

This system is robust in that it can accept transformations from multiple sources at varying rates and be calculated in different ways. Multiple transformation trees can even be connected to form an even more global reference frame containing all the nodes beneath. A limitation is that no

loops or redundancy may be used with this system. To assist with managing these transformations, Robot Operating System (ROS) provides an extensive library to handle all transformation math and management automatically. [2]

1.3 AUGMENTED REALITY

The purpose of augmented reality (AR) is to augment the user's existing vision and view of the world with additional information and added visuals in a way that makes contextual sense. Emphasis is put on the existing view, as without this it not AR, but mixed reality or virtual reality. Paul Milgram describes AR as "Augmenting natural feedback to the operator with simulated cues", meaning whatever simulation is displayed to the user feels natural to them [3]. In another word, AR shows the "invisible" world in the context of the visible world.

Examples of typical "invisible" worlds that can be shown, or reality shifts, are: temporal, spectral, semantic, additive, and subtractive. A temporal shift is showing the visible world at a different point in time. More simply, it shows a history of what has happened. A spectral shift is showing a spectrum that is otherwise outside of a human's visible spectrum, such as a thermal, infrared, or ultra-violet shift. A semantic shift is providing additional information about what is currently viewable, usually in the form of informative labels, with an example shown in Figure 7.

An additive shift is like a semantic shift which adds information to the scene. Unlike a semantic shift, an additive shift does not have to add value to the scene; the scene could be altered to change an objects color, or some 3D model could be added to make it look like its there in the real world. Conversely, a subtractive shift is the removal of parts of the real world. This could be as simple as removing an object on a table. This shift is more typically used to remove something or part of something from the real world, like a wall or the side of a box, in order to show what would normally be hidden by these objects.

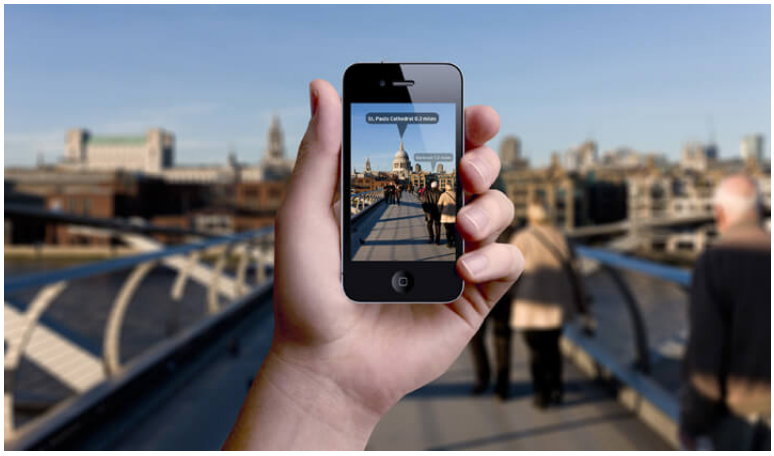


Figure 7. A mobile phone being used as an AR display to provide semantic labels about buildings

In order to perform any of these visual shifts on an augmented reality display, the position of everything in the scene must be known. In order to add a label above a building as in Figure 7, the display must know where that building is located. To be accepted as correct and natural to a user, [4] says the tracked locations of these objects must be within a few milimeters and fractions of a degree. This is a tall order but is necessary when overlaying information and graphics onto a real scene. Section 3 will look into this problem in more detail.

1.4 ROBOT OPERATING SYSTEM

To facilitate our research, we decided to use the Robot Operating System (ROS) framework. ROS is a large set of software libraries and tools that assist with programming robots and other areas of research, particularly computer vision. Because ROS is a large open source project, anybody can contribute useful tools and packages and anybody can use their software for free. ROS also provides a developing environment, automatic message passing between networked machines, can be used in C++, Python, or Java, and has many other features that will be introduced in this section. [5]

The first feature that ROS provides is a developing environment, with the current version named catkin. A typical workspace when using Linux consists of a root directory and a *src* directory that will contain packages for developing. Catkin provides commands for building part or all of the workspace and placing built executables in specified folders that will be automatically generated in the root of the workspace. All the user has to deal with are the packages in the *src* directory. A package is an organization of code or files that serve a common purpose, generally a single component or feature of a robotic system. These packages have files that specify build and export instructions. Using this system allows for efficiently organizing code by functionality.

Another important feature that ROS provides is a common messaging system that allows networked programs to communicate in a uniform way. ROS defines many standard messages that are commonly used, but also supports the generation of custom messages. Using this system, robots and computers using different operating systems can send messages to each other and understand them, and all of this is fully handled by ROS. The networking API allows messages to be sent over topics, which are name-spaces describing a channel to send messages over. These channels can have many publishers and many subscribers, allowing for a scalable networking framework.

Programs written for ROS are known as nodes, where each node generally handles a single task. A node can subscribe to data to use as input and publish data that it generates using the networking API mentioned previously. Nodes can interact with each other from a single machine or many machines, which is particularly useful in multi-robot or multi-device systems. There are also variants to nodes that serve more specific purposes. A common variant is nodelets, which are run on the same machine and avoid wasting network traffic by passing pointers instead of entire messages.

Finally, ROS provides many tools for analyzing and processing data. Rviz is a commonly used visualization tool that can show image streams, depth data, relational positioning of robots and much more. Rqt provides similar visualizations, as well as tools for monitoring network traffic,

including the rates that messages are published at and where exactly that data is going. There is even a data recording tool called rosbag, which can subscribe to a set of topics and record all of the messages sent on that topic with the time sent, along with a few other features. All of these messages go into a bag file that can be used to playback the data at any time, creating a very useful and realistic simulation environment.

There is also an implementation of the transformation tree described in Section 1.2 [2]. This implementation comes as a common library that is easy to use for both putting information on the tree and reading from it. It is also used by visualization tools like rviz to assist with placing objects correctly in the virtual environment.

1.5 GOOGLE PROJECT TANGO

The Google Project Tango is a new prototype smart-phone that has an array of cameras and sensors and does a lot of data processing that we will be utilizing in this project. The device contains a fisheye camera, an RGB-IR camera, an IR projector for depth sensing, and has 2 dedicated graphics processors to perform complex image processing. The Tango device and all of its sensors are shown and labeled in Figure 8.

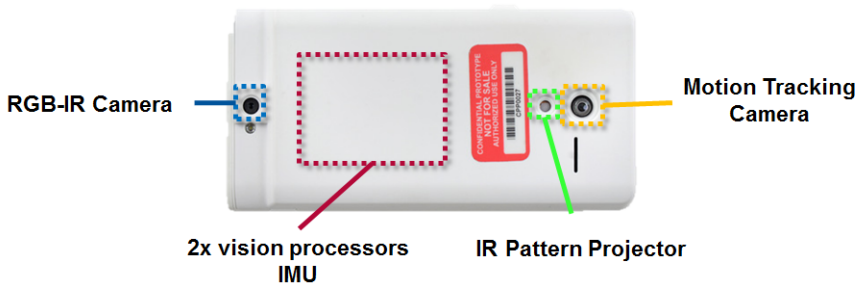


Figure 8. Google Project Tango sensors on the smartphone device.

1.5.1 Tango Pose Estimation

What makes the Tango unique and of great interest to us is its ability to track it's own 6DoF pose, using only its on-board cameras and IMU's. This pose is relative to the point in space where the device was launched. Since the device knows its pose in global space relative to a fixed starting position, it helps with the localization problem discussed in Section 1.3 and Section 1.2.

The Tango is able to to track its own pose by tracking natural features in a scene, and is discussed in more depth in Section 2.1. Specifically, the Tango is performing what is known as visual odometry. In it's most basic form, visual odometry utilizes natural feature tracking to look for very specific and unique features in a scene. The Tango records these natural features over time, and based on how these features move around frame to frame the Tango can make a guess as to what its own pose is in the scene. To get as many features for tracking as it can, the Tango

uses its fisheye lens for feature tracking which gives it a 120 degree field of view. The prototype Tango device is reported to have only 1% of drift error over length of path driven, providing a very strong camera localization solution.

1.5.2 Tango Global Coordinate Frame

The Tango device defines its global coordinate frame to be the position when the Tango application is first launched, with the device oriented in landscape mode. It uses the IMU to position its positive Z axis to be pointing up against gravity, leaving the X and Y axis to be perpendicular to the ground plane in right handed fashion. The positive Y axis is facing in front of the tango device itself and the X axis to the right of the phone. This is different from the convention that ROS uses as its world frame in the RViz visualization tool, where ROS uses x as being forward.

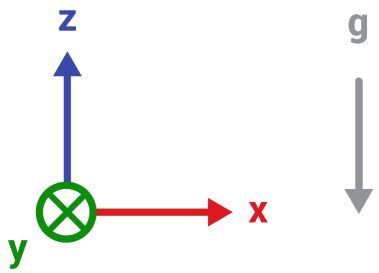


Figure 9. The global coordinate frame of the tango, with respect to gravity

1.5.3 Tango Depth Readings

The Tango also comes with an IR-RGB camera and IR projector which allow it to measure depth in a scene. Using the same concept as the Microsoft Kinect sensor, the Tango projects an IR dot pattern over everything in front of it and then uses the IR-RGB camera to see this dot pattern. It can then determine how far away objects are based on how the dots lie on surfaces in a scene. An example of the dot pattern projected on a persons hand is shown in Figure 10.

1.5.4 Tango Limitations

The Tango is still in the early prototype stage and has a number of limitations in terms of usability. The major downfall of the device is its battery life and overheating problems. The battery itself will last for about an hour or two at most. This is not terrible for all the computations it is doing on the GPU, but is not ideal for a marketable and potentially consumer device.

The real problem is that Tango has a tendency to fail and crash the application after heavy extended use of the pose estimator, and overheats the device. This requires a lengthy cool down time where the Tango can not be used because it will either give inaccurate estimations or continue to fail and crash the application again. In extreme cases of overheating, the device will actually shut down completely and must be rebooted in addition to waiting for it to cool off.

Figure 10. IR dot pattern projected onto a scene of a body and hand

All of this is currently understandable since the device is still a work in progress at Google and will continue to improve until its final release. The final version will be in a tablet form factor, and will have great improvements to power consumption and pose quality.

1.5.5 Our Use for the Tango

As described, the Tango is an amazingly capable device that gives access to a multitude of useful data streams that is not available in any other single compact device. With that being said, it is the perfect sensor package for the needs of this project. Namely, we will be using the tango device to pull 2 data sources for processing. The first is its own 6DOF pose that is relative to its starting point, which gives us a global root frame. The second piece of data we need is the image data coming from the IR-RGB camera. This will be the image stream we use heavily for image processing in this project.

1.5.6 Intrinsic Parameters of the Tango Cameras

Camera intrinsic parameters are used to describe the relation between individual pixels from a camera imager and real space. They also describe the focal length of the lens as well as distortion coefficients, all of which are used during the image rectification process which simply straightens out a scene making it easier for computer vision algorithms. We use these to estimate 3D positions of objects found in images.

Having high quality intrinsic parameters is essential when relating image space to 3D space. These parameters can be calculated from information in the data sheet for the camera, but this was not available to us. Our next option is to use a calibration tool in ROS that has us hold a checkerboard of known dimensions in front of the camera while the software estimates the intrinsic parameters. In our past experience, the results of this tool have not been very good. We chose the final option, which is to use the intrinsic parameters provided with the Tango from the factory, despite warnings that the intrinsic parameters from device to device may be different.

2. BACKGROUND

The main problem addressed by this is estimating the poses of objects in 3D space. [6] discusses the significance of properly tracking people when trying to duplicate human motion. The motions of over 100 joints need to be tracked in order to create a realistic model. [7] demonstrates the importance of having accurate pose estimates when creating an augmented reality display. The spatial relationship from the display to the object must be correct in order for anything on the display to correctly correspond to real world objects.

In Sections 2.1 and 2.2 we discuss different methods of tracking objects. Section 2.3 illustrates ways of estimating the pose of a camera for the purpose of assisting an augmented reality display. Finally, we considered the connections between object tracking, augmented reality, and human robot interaction, discussed in Section 2.4.

2.1 NATURAL FEATURE TRACKING

Knowing the positions and orientations of objects is a common problem in many different fields, most commonly in robotics and augmented reality. Common implementations of object tracking are done by looking for fiducial markers that can be easily seen in an image and are placed on the object itself, which is further discussed in Section 2.2. A much harder method still being researched is tracking the natural features of objects. The theory is that any object could be tracked with no prior knowledge, or additional markers, based on features it already has.

One of the simpler implementations uses the edges of objects as track-able features. Edge-based detection methods are great because they are generally easy to implement, are computationally efficient, and can handle changing lighting conditions [8]. When the lighting changes, the images captured by a camera will look different, even if nothing in the scene moved. Therefore, it is critical for vision based trackers to handle changing lighting conditions because they require light in order to function. If a system is tuned to a particular amount of ambient light, it will not function properly when that amount of light changes. Properties like color or reflectance can change slightly with lighting conditions. They handle this by searching for high contrast features in an image, which show up well whether there is a little or a lot of light.

When performing edge tracking, an accurate 3D model of the object is needed beforehand [8, 9]. One implementation of this method is to search the 3D model for similar edges to those detected in an image. This technique is usually limited to simplistic shapes with many straight lines. With this constraint, the complexity can be reduced, allowing for easier matching of similarities. This solution does not work well with occlusion, which occurs when parts of an object are hidden from a camera [9].

The previous methods are subject to problems like occlusion, which occur when only a portion of a tracked object can be seen because another object is in the way. Complex objects often have this problem because they cannot be completely seen from any point of view. An alternative tracking method was proposed by [9] that allows both for occlusion and the tracking of more complex objects. Any object can be tracked, assuming a 3D model is present and an initial guess in pose is given. In

this case, they used their previous work [10], which does not work under occlusion, for the initial guess. Once an initial guess is made, [9] uses a particle filter to render different projections of where the object could be, and performs an edge filter to produce an image of just the edges. The same edge filter is done on the real input image. Instead of using the lines to match, they instead take each edge filtered image from the particle filter, and perform a bitwise AND on the input image to find overlap in the pixels from each image.

This method does however require a priori knowledge of the shape of the object, namely an entire 3D model of the object. This limits the system as it must keep a database of every object that it wants to track. Edge based methods are also greatly susceptible to cluttered backgrounds and noise [8].

Non-edge and non-3D model based trackers are much more difficult and complex. This is typically implemented by extracting features from the scene. These are features the system finds unique and can find again when looking at the history of where that feature is [8, 11]. Optical flow can then be used to calculate the velocity of each feature. This method is also known as visual odometry and is discussed in more detail in section 2.3

2.2 FIDUCIAL TRACKING

When working with objects that that do not deform, it is often simpler to place fiducial markers onto that object to assist with tracking than it is to develop a new tracking algorithm. These fiducials can be optimized for tracking, like the IR reflective markers used in motion capture rooms, Figure 4, or like the AR tags in Figure 11. Instead of having to track any number of objects in a large database of 3D models, a much simpler tracking algorithm may be written to only search for reflective fiducials, which ideally show up as small white dots on gray-scale or IR images. AR tags may be tracked as a rigid set of lines and intersections on a single plane. While they contain natural features, AR tags are considered fiducials because they are added to a scene to provide additional localization references.

When tracking reflective fiducials, there are two additional steps in the process after finding them; identifying groups that make up objects and estimating the poses of those objects. It can be a difficult process matching a large set of tracked fiducials to one or more objects that could appear in a scene. Libraries like OpenCV [12] and PointCloudLibrary (PCL) [13] provide implementations of PnP, RANSAC and many more algorithms, which can be used to solve for sets of fiducials.

In robotics research, needing to know the pose of objects is a common problem, especially in the realm of manipulation. [14] utilizes a well known AR tag tracking package, AR Toolkit [15], to do their tracking so that they may move their robotic arm to follow the movements of their tracked object. They started by placing AR tags on each side of a cube to provide the pose of the cube relative to the camera. As long as the camera is pointed at the cube, ARToolkit will be able to calculate the pose of the cube based which AR tag is visible and what its orientation is. Because of its ease of use, ARToolkit has become widely used for tracking purposes [8].

Sometimes reflective markers or AR tags are not well suited for a particular application because of constraints like lighting or changing nature of objects that need to be tracked. In these

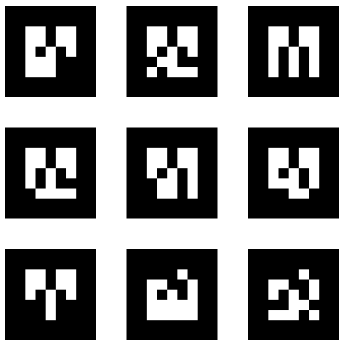


Figure 11. AR tags that look and work similar to Quick Response Codes (QR codes).

cases, simple colored fiducials may be used to identify objects. [16] uses red and blue tape on the user's index finger and thumb respectively to assist with tracking. Their algorithm searches for portions of the image that contain large enough chunks of red and blue pixels, calculates the centroids of these areas and uses those as centroids for gesture recognition. Normal hand tracking, which searches for the shape or color of a typical hand, was not used because the goal was to track a surgeon's hand over a patient's abdomen while planning for surgery. The red and blue tape provided fiducials that contrasted the background in nearly all cases and will also work with different hand sizes, as long as enough tape is used.

2.3 TRACKING CAMERA POSE

Section 2.1 describes methods of tracking natural features to find objects in a scene to estimate their poses. Sets of natural features typically make up objects. When tracked in consecutive image frames, these sets can used to estimate camera movement with relation to previous frames.

Similar features can be found in AR tags or other fiducials placed in a scene for visual localization purposes. [17] elaborates on an algorithm used to solve for the pose of a camera based on 4 points found on a planar object in a known geometry. Such a process can solve the camera localization problem in a global frame if the location of the object is known in advance using the technique in Section 1.2. If the object's location is not known in advance, but is tracked over time, the camera's relative pose can be tracked instead.

A method for tracking the camera's pose using only images is described in [11]. Here natural features are tracked from frame to frame and referenced against two spatially separated reference images taken at known locations. The pose of the camera is estimated based on the displacement of natural features from the current frame against the natural features in the reference frames. This process yields a camera pose relative to those reference frames. A major advantage of this system is there is very little drift because all poses are calculated directly from fixed reference points, so the pose does not accumulate error over time. [11] reports success within a small area and constrained range of motion as well as the possibility for scaling to large environments by adding reference images with known displacements between the camera positions. Unfortunately, this will not work with changing environments nor is it possible to have reference images for all possible environments.

In [18], multiple means of estimating the pose of the camera are considered, but ultimately GPS localization is chosen because the application in mind is used to show AR over entire buildings. Pose data is supplemented using a digital compass and gyroscopic sensor present in modern mobile tablet and smart phone devices. While the solution proposed in [18] is well suited for large scale environments, GPS measurements will typically have 3 or more meters of error, which is unacceptable for interactions with robots, dynamic environments or any environment with objects smaller than buildings.

2.4 HUMAN ROBOT INTERACTION

An increasingly popular application for AR technologies and devices is in the robotics field. AR research is being done to look at how humans can best collaborate with robots through human robot interaction (HRI). Instead of robots being used solely in closed industrial settings and programmed by experts, they are now being used as companions to the human workers.

In order for robots and humans to collaborate more, there are more demands on the robotic and AR systems. Companion robots must be easy to understand and control, as well as being safe to work with [19]. An untrained human operator must be able to see and understand the robot's data, and communicate his or her input back to the robot. Robot data is inherently difficult to understand, and the best way to understand it is to present it to the user in a way that is visual and more importantly contextual [14, 19]. Augmented reality devices can be used to bridge the gap between robot data and human understanding by visualizing that robot data in a way that makes spatial contextual sense to the human user.

Typical AR devices used for visualizing robot data can range from projectors on the robot itself, mobile monitors such as smartphones, or head mounted displays that have semi-transparent displays. With these devices, additional virtual information can be rendered on the scene such that the virtual information and physical scene co-exist [19].

One use for these AR devices in robotics is visualizing a robot's path. This is a heavily researched topic in HRI, as a human would want to be able see what path a robot plans on taking and then be able to modify the path. By using a monitor based display, the 3D path and other virtual additions can be visualized by correctly registering the path in 3D over the scene taken from a camera [14, 20]. [19] implemented a similar setup with a projector mounted on a robot's end effector, which is used to visualize the robot's 2D path across a surface without the human having to wear or hold additional devices.

After a path is visualized in the correct spatial context to the user, the user must be able to intervene and correct the path. The solution of [14] is to use a hand held cube with an AR marker on each side which can be tracked in 3D space. With this, the user can point the cube at points in the path, which is being overlaid onto their AR screen. This will allow them to delete or move the points in the path. In the case of the 2D path across a surface in [19], a tracked pen is used to redraw the path on top of the surface that is being projected on. The user drawn path is then the path that the robot will now take.

3. MOTIVATION

Section 3.1 illustrates the need for a localization solution with augmented reality devices. Section 3.2 shows how our current motion capture lab is not sufficient for ideal use.

3.1 AUGMENTED REALITY AND LOCALIZATION

Augmented Reality is rapidly growing in research due to the many applications it could potentially be used for. In most applications of AR, the purpose is to provide the user with some more information about what they are currently looking at with their device; this could be a phone, tablet, head mounted display, or even a watch. One major concern with augmented reality displays is where this extra information comes from.

One large source of information can be and all of their sensors. Robots have a lot of raw and registered data that makes sense to a robot but does not necessarily translate well into what a human can understand. The motivation behind AR technologies for us is to take in these raw data sources from robots, make further sense of that information, and then finally present it to the user in a way that makes visual and contextual sense through an AR device.

The scenario that we wanted to look into specifically can be best illustrated by Figure 12. In this figure there is human user holding an AR tablet display, with a companion robot on the other side of a wall. Under normal circumstances the user has no way of knowing what is on the opposite side of the wall, but the robot on the other side is equipped with many cameras and sensors. All of these sensor streams can be fed back to the user so they can see what the robot is seeing. In this case, they could see that a person exists behind the wall.

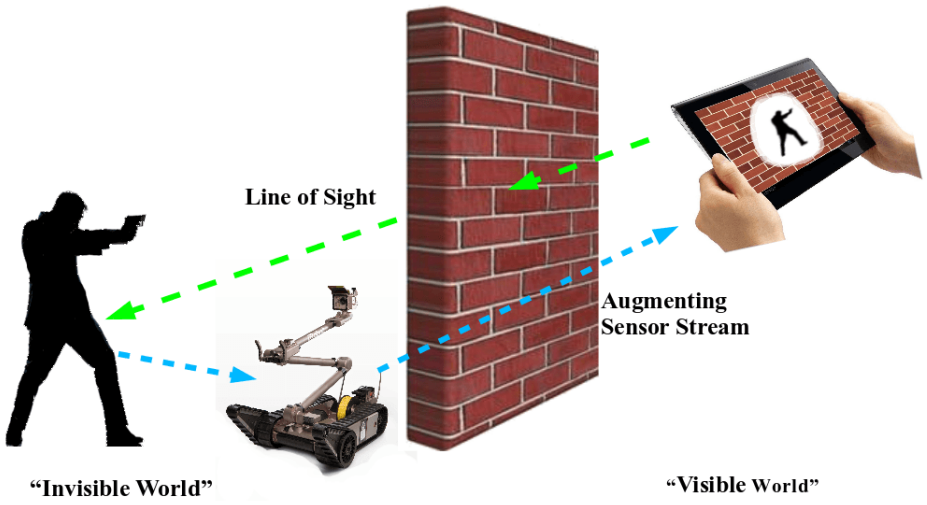


Figure 12. An augmented reality system showing the user an object on the other side of a wall.

For this data to make the most sense to the user, it should only ever be displayed when it makes sense to. When it does make sense to display it, it must be positioned where it would be spatially expected. If the AR tablet is thought of as a window and is held up while looking at the wall, only then should information about what is behind the wall be presented. This is an example of a subtractive shift that was discussed in Section 1.3.

In order for this data to be presented spatially and contextually correct, the 6 DoF pose of every object has to be tracked. This includes the robot itself, the tablet display, and ideally the gaze of the user. The extra information must be rendered on the tablet or any AR device as if it co-existed in the physical world. The accuracy of each tracked object must be high enough so the information is exactly where it should be, within a few millimeters and fractions of a degree [4]. The gaze of the user helps to enhance this effect even more so they are projected information in exactly the direction and angle they are looking at the display.

3.2 PROBLEM WITH CURRENT MOTION CAPTURE

The current solution for localization used by MITLL is a motion capture lab. Currently, the system is calibrated to produce pose estimates with about 1 mm of error in position. This is an excellent solution to our localization problems when we are towards the center of our 37 square meter capture area, which also reaches up about 3.25 meters.

Unfortunately, coverage is very poor close to any of the walls or the entrance, which is a large opening in a hanging net. Moving tracked objects towards these areas gives intermittent readings if the object isn't lost completely. This is due to the reflective markers on the object leaving the capture volume, being occluded from cameras, or in some cases being behind them.

Because the motion capture system was not designed to continue tracking objects that move behind cameras or are fully occluded, we will only consider the possibility of expanding size of the capture space. The first consideration is the space itself; our room is too small for the number of cameras in it. We are currently using 20 cameras, which could cover over 80 square meters if spaced out optimally. To set aside this much space for a permanent motion capture lab is not currently feasible.

The second option is to place cameras elsewhere in the lab without dedicating space, some static to mount on other walls and some on stands so that they can be reconfigured when space is needed. This can cost from $1000 to $2000 per camera in addition to having a host computer capable of handling all the processing. Covering hundreds or even thousands of square meters would be very expensive. The reconfigurable space would cost less, but a 30 to 60 minute calibration procedure must be performed when any camera is moved. Again, this does not scale well when considering large changing environments.

4. PROBLEM STATEMENT & GOALS

A reasonable solution to overcome the spatial constraints of our current motion capture system, described in Section 3.2, would be a mobile system that could be started up with minimal calibration. An ideal mobile system would consist of just one camera, or sensor package, which would do the sensing of a typical motion capture system. Total capture volume at any point in time could be traded for a smaller sensor package that might not have a wide field of view. With these constraints so far, the new system would be limited in how many objects it can track at once, how occluded those objects may be, and how accurate pose estimates will be.

A simple robotic application is also presented, with the idea of demonstrating how this tracking system could be used in practice. We looked to use a Turtlebot mobile robot as our robotic platform, with a pan and tilt turret that would hold our mobile motion capture system. With the Turtlebot and turret, we present a mobile robot capable of following a tracked object using only our motion tracking system for sensing.

4.1 PERFORMANCE METRICS AND GOALS

Table 1 shows all of our minimum system specifications as well as reach goals. These goals are based on what is necessary for a good AR device as well as specifications for our current motion capture system. The items in this table are ordered by importance, with the most important at the top. Testing procedures and calculations will be explained Sections 5.4 and 5.5.

TABLE 1

Minimum expectations and reach goals in various categories.

Category	**Minimum**	**Reach Goal**
Position Accuracy	1.00 cm of error	0.50 cm of error
Orientation Accuracy	5 degrees of error	2.5 degrees of error
Moving Objects	Tango & tracked object moving	Tango mounted on a robot
Frames Tracked	90%	97%
Operating Speed	10 Hz	30 Hz
Object Distance	0.5 to 2 meters	0.5 to 3 meters
Object Rotation	15 degrees	60 degrees
Number of Object	1	4

Values for the minimum orientation and position accuracy are roughly $1/10th$ the accuracy of our current motion capture system. The minimum operating speed of 10 Hz is also close to $1/10th$ of the maximum operating speed, 120 Hz, of our current system. The actual operating speed of our system will be somewhat dependent on the percentage of the frames tracked because our system has to be able to process its data and track any objects for a high percentage of the frames to

earn and operating speed of 10 or 30 Hz. These are the most important specifications because we intent to use our system with an AR display, which requires a generally high accuracy in its pose estimates at a high enough rate to make visualizations that move slowly.

The most important quality that cannot be measured with numbers is the ability to have the device and tracked objects moving simultaneously and not necessarily synchronously. This goal can by broken into many small goals, like performing the tracking with all objects stationary, perform the tracking with a consistent background, etc. Fortunately, having either the Tango device or the tracked object moving are very similar problems, so there may be shortcuts in our list of intermediate steps. However, mobility is our solution to the limitations of current motion capture technology as well as our major addition to the AR display. Therefore, we must achieve a fully mobile system.

While it is paramount that our system is fully mobile, we have roughly defined reasonable motion so that we may guarantee our final product works under certain conditions. Reasonable movement shall be slow to regular walking pace for a human, without jumping or erratic twisting of the body. This will allow us to track an AR display or from an AR display without any trouble, but would rule out tracking quad rotors or other small unmanned aerial vehicles reliably. Robots in our lab that could be used in such a system are the Kobuki Turtlebot or iRobot Packbot, which move at speeds fairly close to a person's walking pace.

The object distance ranges are estimated with respect to typical industry depth sensors, like the Xbox Kinect, which have been thoroughly characterized to have an effective range of 0.8m to 4.0m [21]. While the depth sensor on our Tango device has not been fully characterized and studied, we performed a basic test to estimate the distance error. After these tests we decided to not use the depth data from the Tango, but still to maintain the same expectations effective distance from the sensor. Our depth sensor characterization strategy is defined in Section 5.4.5 and our additional findings are in Section 6.5.

The final categories are object rotation and number of objects. These are a measure of usefulness based on our experience with the AR display. The object rotation is measured with respect to the Tango device and the minimum is quite small because we expect that this will be a difficult challenge to solve using one camera as we intend to. Similarly, the number of objects to track is low for the same reason. However, with these minimum goals reached, our system will fulfill its purpose of expanding our motion capture space and assist the AR display.

5. METHODOLOGY

The following section is our design, implementation, experimentation and testing methodology. Here we highlight the important theory for the various components of our mobile motion capture system as well as major concerns that arose during implementation.

For all of our development, we started with a general plan then implemented the bare minimum of each component in the system for an initial proof of concept. When necessary, we revisited components to iteratively improve them to what they needed to be. All of our code is currently written in Python, making what we consider to be the prototype version of our system.

Sections 5.1 and 5.2 describe some of the planning we did before diving into the implementation of our mobile motion capture software described in Sections 5.6 to 5.10. Sections 5.3 to 5.5 describe the hardware and software used for experimentation and testing of our system as well as the procedures followed. Section 5.11 describes the extension to our project that demonstrates how the mobile motion capture system can be used in a robotic application, where we have a mobile robot follow a tracked object.

5.1 SYSTEM ARCHITECTURE

Based on the system requirements defined in Section 4.1, we designed a minimalistic system architecture and data flow to accomplish our goals, provided each individual component adheres to overall system constraints. Figure 13 shows this system as a simple pipeline from one major process to the next.

The first block at the top is the Google Project Tango device, which outputs a luminosity image and its own pose, see Section 5.6. All of this data is sent into the ROS environment where it can be accessed by the software we implement as well as visualization and data logging tools. Specifically, we intend to collect this data using rosbag, a ROS data recording and playback tool [22]. Through the ROS environment, either rosbag or the Tango device can be the source of data for our software, allowing us to perform on-line and off-line system tests.

The yellow block is the fiducial tracking system, which will be explained more in Section 5.7. It takes in luminosity images, filters them and searches for fiducial markers. These markers will show up as bright white dots. For each image searched, a FiducialArray message will be sent to the next major system, which is the Pose Estimation system, explained fully in Section 5.10. In this subsystem, constellations are identified from arbitrary points, then their poses are estimated and piped out as Poses using the ROS TF library. An additional component to the pose estimation system is an external database of known constellations used as reference when searching for constellations and estimating poses.

5.2 DATA TYPE DEFINITIONS

In order to organize our data and provide common data structures for the various parts of our system, we created the data types defined in Sections 5.2.1 to 5.2.4. Having these data types allowed

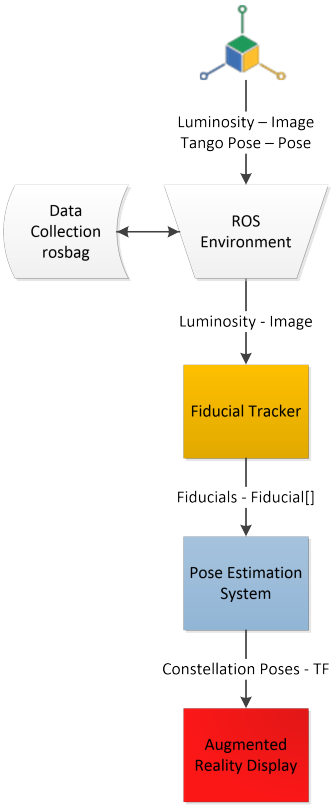


Figure 13. System overview of the major components in the Mobile Motion Capture pipeline.

us to write separate nodes for each part of our system with a common means of communication. All of these message definitions are located in the *mobile_mocap_messages* package and are compiled using the ROS message generation tools introduced in Section 1.4.

5.2.1 Fiducial Message

In order to represent fiducials, we created this Fiducial message with the data listed in Table 2. The x, y and z fields are used to represent the 3D coordinates of a fiducial that is either known or being tracked. We also use these fields to represent a 2D image coordinate coordinate, with the z field set to a default value that represents that the point is on an image plane instead of in 3D space. The assumed coordinate system for these points is as follows: z is the distance from the sensor, the positive x axis goes right and the positive y axis goes down when looking from the sensor.

TABLE 2

Fields in a Fiducial message.

Type	Field Name
float32	x
float32	y
float32	z
float32	size
float32	uncertainty

The *size* and *uncertainty* are extra fields used to hold extra data calculated during the fiducial tracking process. The *size* field typically represents the size of the fiducials in pixels, which will vary at different distances, fiducial types and noise. The *uncertainty* field can also be set during the fiducial tracking process and can show how much error may be associated with a particular fiducial.

This message definition was chosen instead of the existing Point message because we wanted to be able to add extra fields as needed without having to modify code that depended on receiving a particular message type. Standard ROS messages would not give us that flexibility because we cannot modify them.

5.2.2 SubConstellation Message

SubConstellation messages are nearly identical to Fiducial messages because they each are capable of representing 2D or 3D points along with uncertainty. The main reason for creating this message is what it represents; this message holds the centroid of a sub-constellation, which is a linear group of fiducials. The *name* field is used to identify the sub-constellation once identified. In addition to the centroid and name, a SubConstellation also holds all of the Fiducials as a Fidu-

cialArray that make up the sub-constellation. Table 3 below shows the fields in a SubConstellation message.

TABLE 3

Fields in a SubConstellation message.

Type	Field Name
string	name
float32	Cx
float32	Cy
float32	Cz
float32	uncertainty
FiducialArray	fiducial_array

5.2.3 Constellation Message

For representing groups of fiducials as constellations, we made the Constellation message with the fields listed in Table 4. A Constellation also as a unique *name* identifying it, *point_names* and *sub_constellations* that make up the constellation, as well as some tolerance values. For example, *xyz_tol* specifies the tolerance in measurement that any fiducial can have in each dimension. Other possible fields could provide similar tolerance information useful when trying to fit a constellation that may be need a high tolerance in order to be detected.

TABLE 4

Fields in a Constellation message.

Type	Field Name
string	name
string[]	point_names
SubConstellation[]	sub_constellations
float32	xyz_tol

This message format was chosen instead of the existing ROS PointCloud2 message because we wanted to have the same level of control over our messages discussed in Section 5.2.1. We also wanted to have an easier format to work with than PointCloud2 messages, which have to be parsed in application specific ways from a byte array, which is not an issue with the Constellation message because the fields are clearly labeled.

5.2.4 Array Messages

We defined the FiducialArray and the ConstellationArray messages for sending groups of fiducials and constellations respectively. This is used for grouping messages common to a particular sensor frame at a specific point in time. An example would be the group of all fiducials found in a single image frame. In this case, the *header* from that Image message would be copied into the FiducialArray to preserve the time *stamp* and $frame\_id$ that identify the time of the reading and sensor origin.

Table 5 shows the fields for a FiducialArray and Table 6 shows the fields for a ConstellationArray. The $fiducials$ field is an unordered, variable length list of Fiducial messages representing fiducials found at the time specified in the header. The *constellations* field is the same as the $fiducials$ field, except it holds Constellations.

TABLE 5

Fields for a message used to send groups of fiducials common to a particular time and sensor origin.

Type	Field Name
Header	header
Fiducial[]	fiducials

TABLE 6

Fields for a message used to send groups of Constellations common to a particular time and sensor origin.

Type	Field Name
Header	header
Constellation[]	constellations

5.3 HARDWARE

In this section we discuss the various forms of hardware we utilized and designed to work alongside the software we developed. The Tango device on its own has an RGB-IR camera, and we discuss our addition of an IR flood light and IR filter in Section 5.3.1. The most important hardware is the mounting equipment developed described in section 5.3.2, which allows for tripod use, and more importantly a consistent and repeatable data collection procedure.

In addition to the mounting hardware, we utilized the motion capture lab, described in Section 1.1, as well test constellations, and an additional computer. Section 5.3.3 describes these test constellations, and the computer we used for off board and post-processing is detailed in Section 5.3.4. Sections 5.3.5 and 5.3.6 list the parts used in our robot following application.

5.3.1 Infrared Hardware

The Tango device has an RGB-IR camera, which means it works 2-fold as an RGB and an IR camera, where a fraction of the pixels are dedicated for IR light. In order see the reflective fiducials well, an IR flood light was purchased to be used alongside the Tango in order to flood the scene in front of it with IR light; this bounces the IR light off the fiducials into the camera. A small IR filter was also purchased in order to be placed in front of the Tango's RGB-IR camera so as to block all visible light from reaching the sensor, since the RGB portion of the image is of no use to us.

5.3.2 Tango Mount

In order to have controlled experiments, we needed to not only keep our Tango device and test constellations stationary, but also have accurate pose estimates of each for verification against our system later. We used the motion capture lab for all measurements and Section 5.3.3 describes specifically how we tracked the constellations. Because having a person simply hold the smart phone cannot keep it perfectly still with an additional IR light on the side, we constructed a rigid frame to hold all of these components.

Figure 14. The original 3D printed Tango holder designed by [23].

We started with a 3D printed handle from [23] for holding our Tango device, shown printed in red in Figures 14 and 20. This was a great starting point because the very first print was able to hold the Tango smart phone snug and secure without any modifications. Unfortunately, this part

didn't have any means of mounting other things, like the IR light or a tripod. Our solution was to copy the geometries from the existing Tango handle, remove the handle portion, and add a hot shoe mount at the bottom. Before deciding to use hot shoe mount, we considered having a tapped hole for a screw or inserting and gluing a bolt into a hole in the bottom to mount the device on other standard camera related mounting equipment. We decided on the hot shoe mount instead because it could be used more easily with the mounting equipment described later in this section and the simplicity of fabrication.

Figure 14 shows the start of our design and Figure 15 shows the final version of the printed portion. Because of imperfections in our first prints, we had to revise the model a few times to reduce the amount of overhang in the portion that actually wraps around the smart phone. This part was designed in SolidWorks and the final version was printed using ABS plastic on a MakerBot 3D printer. See Figure A.1 for the final dimensions of the tango holder.

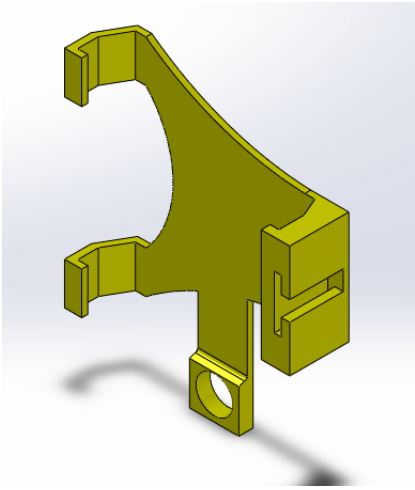


Figure 15. The final version of the 3D printed Tango holder.

The next thing to mount was an extra IR light source, whose necessity is explained in Section 5.3.1. In order to suit our needs, this light had to provide light at about 850 nm, which is in the near infrared spectrum, have its own power source, be small enough to carry with the phone and big enough to provide enough light. For this we considered wiring our own IR LEDs, but could not make such custom electronics due to time constraints. Current products on the market that could suit our needs fell into two categories: closed circuit television (CCTV) night vision illuminators and IR night lights for digital cameras and camcorders. While the CCTV illuminators were close, they actually required large power sources that would not be practical, so we decided on a IR night light for cameras. Because these are designed for cameras, they already have hot shoe mounts, are light weight and are powered by batteries. The exact IR light that we used was a Sima SL-100IR Infrared LED Light for camcorders with night vision.

For tracking the Tango device, we used reflective fiducial balls, which work best in the motion capture lab. We attached them by cutting two plastic fiducial mounts to fit in the extra hot shoe

mounts on our IR light and one final marker with velcro on part of the 3D printed part that wouldn't obscure the Tango device's cameras.

The rest of the mount consists of purchased parts, listed in Table 7. The Tango holder is the only part that we printed and the rest of the items were acquired through Lincoln Labs. All parts have 1/4-20 UNC threads, hot shoe male or female mounts, or both for mounting to any other part. A late addition to the mount is the Thorlabs IR filter that only lets IR light through. It is one half inch in diameter to fit exactly over the RBG-IR camera on the Tango smart phone. It is held in place in a hole in the 3D printed holder with a small dot of glue.

TABLE 7

Components of the Tango holder and mounting system

Item	**Description**
Tango Holder	3D printed Tango phone holder
Rotolight Foam Hand Grip	A handle with standard camera mounting studs
Sima SL-100IR Infrared LED Light	Projects additional IR light for the tango
Tripod	Standard camera tripod
Revo Hot Shoe Adapters	Hot shoe to 1/4"-20 stud adapters
Thorlabs IR Filter	An optical filter for 850nm light
Assorted Reflective Fiducials	Motion capture lab reflective fiducials

Our final version is shown in Figure 16 with most of the parts from Table 7. In grey is the 3D printed Tango holder, which was printed in grey ABS plastic. The fiducials are circled in red and were placed in these locations so that they will be visible as much as possible when in the motion capture lab. The blue arrow points to the assorted camera mounting hardware. The green arrow points to the IR light, the purple arrow points to the Tango and the yellow arrow points to the IR filter.

This Tango mount, handle and IR light source are intended to be the final version of the mobile motion capture system, unless a Tango tablet arrives. A new 3D mount would need to be designed for the Tango tablet, possibly without the handle and with an entirely different array of IR LEDs due to the different shape of the tablet.

5.3.3 Test Constellations

In order to test different fiducials in varying configurations, we placed reflective stickers on a few poster boards in both linear and non-linear test patterns. We started with linear test patterns to satisfy our initial implementation described in Section 5.9. To make these patterns, we used a straight edge to draw a straight line on the poster board, measured out distances between stickers, then placed and labeled them. To correct for human error in measurement, we used the motion capture lab to give more precise measurements of the distances between fiducials. We used these

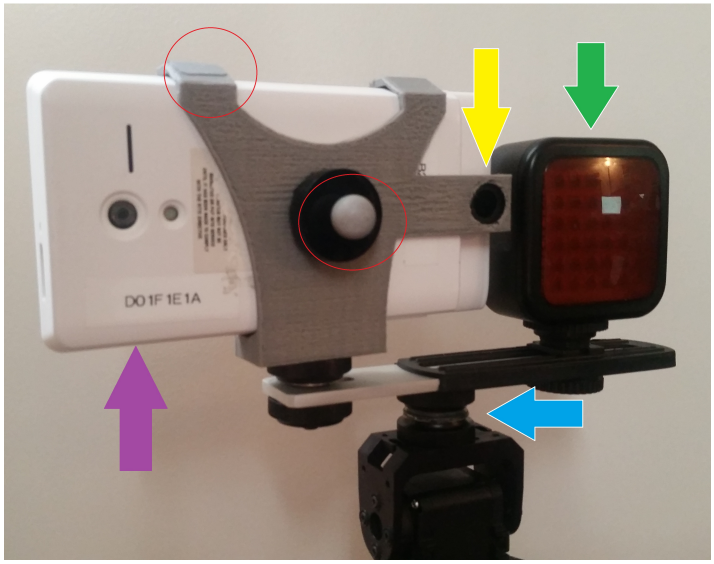
tango.jpg 

Figure 16. The final version of the Tango holder with components annotated.

poster boards to test our systems ability to tell linear constellations apart, identify constellations at different distances and orientations, as well as test our fiducial tracking system against noise, distance, movement and different types of fiducials. Figure 17 shows a single simple constellation of circular stickers on a poster board.

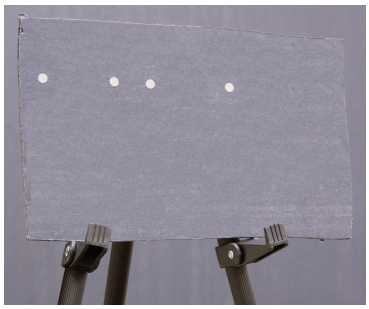


Figure 17. A simple linear constellation of circular fiducials.

Table 8 shows the linear constellations that we placed on poster boards for testing. Patterns A and B were chosen for the initial testing of fiducial tracking. These, and patterns C through F were used for comparing how well we could tell constellations apart because of how small their differences in measurements are.

TABLE 8

Description of each linear test pattern used during testing. The 2nd, 3rd and 4th sticker values are distances in mm from the first sticker along the same straight line.

ID	Sticker Type	2nd Sticker	3rd Sticker	4th Sticker
A	Square	83.07	136.91	219.27
B	Square	65	155	235
C	Circle	25	50	75
D	Circle	50	90	150
E	Circle	35	81	121
F	Circle	65	98	148

5.3.4 Additional Computing Power

In order to handle the image processing software used in Sections 5.7, 5.9 and 5.10 as well as data logging, we used an additional laptop computer to handle all of this. The operating system is Ubuntu 12.04 with ROS Hydro installed, both are long term support versions. The CPU is an Intel Core i7 and the GPU is an NVIDIA GeForce 610M, providing plenty of computing power for handling the ROS environment, communication with the Tango smart phone, and additional tools for monitoring the program state.

5.3.5 Mobile Robot

For the purposes of mobilizing our mobile motion capture system, we chose the Kobuki Turtlebot 2, Figure 18 as our robotic platform. This simple robot has two motorized wheels for driving as well as casters for balance allowing it three degrees of freedom. In general, turtlebots are capabale of driving straight forward or backwards, rotating in place or driving in arcs.

The turtlebot is a complete robot that only needs assembling. To actually run it, there are many ROS [5] programs specifically written for handling the sensors and motors on the robot. With all of this provided for us, we just have to send specific ROS messages describing the velocity that we want the robot to move, and the provided programs will handle that for us.

Additionally, the base of the Kobuki Turtlebot 2 provides multiple 12v and 5v power sources, some of which will be needed by the servos described in Section 5.3.6.

5.3.6 Pan & Tilt Mechanism

In order to allow really precise tracking from a mobile robot, we decided to install a pan and tilt mechanism to orient the Tango sensor on top of a mobile robot. Because we planned for this implementation to be a prototype, the only requirements we had for this device were the following:

1. Must be strong enough to move the Tango device

Figure 18. Kobuki Turtlebot 2.

2. Must have reasonable precision
3. Must have large range of operation
4. Must be able to construct the hardware quickly.

Because of these requirements, we decided to purchase a mechanism rather than design our own. Our main concern was getting a system with servos strong enough to move the Tango and other hardware from Section 5.3.2, which has a mass of roughly $0.5kg$. We estimated that the center of this mass is $5cm$ from the mounting screw and that the mounting screw would be withing $10cm$ of the center of rotation for a given servo used for tilting. With these estimates, the minimum approximate torque we would need is as follows: $(0.5kg) * (5cm + 10cm) = 7.5kg * cm$. Through the Trossen Robotics website [24], we found a pan and tilt servo kit that used Dynamixel AX-12A servos, which provide $15.3kg * cm$ of stall torque, which provides a good safety margin for our purposes. Additionally, these servos have a 300 degree range of motion and 0.29 degree resolution, meeting all of the first three requirements. The final requirement was met by the kit itself, which we purchased and assembled without any difficulty.

We also purchased the parts described in Table 9 to assist with the wiring and interfacing with the servos. The T connector was chosen as the common connector between all power sources and adapters. An additonal wire was made to provide power from the Turtlebot and interface via a T connector.

Figure 19 shows a high level diagram of how to wiring multiple servos. Even though these servos are connected sequentially, they can be controlled independently as described in Section 5.11.1.

TABLE 9

Parts for the pan & tilt mechanism purchased from Trossen Robotics.

Item #	Name	Description
KIT-PANTILT-AX	PhantomX Pan Tilt	Pan and Tilt Servo Kit
RO-902-0032-000	Robotis USB2Dynamixel Adapter	Adapter for USB to servo control
POW-12v5a	Power Supply 12V - 5A	Wall outlet power adapter
C-200-925	T Connector Male-Female Pair	T connectors

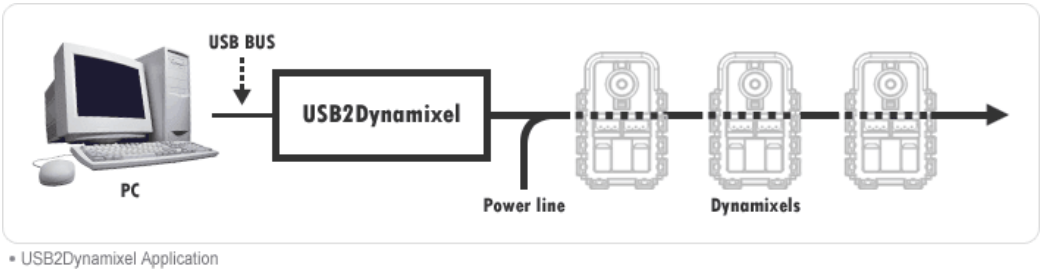


Figure 19. Control circuit for Dynamixel servos [24].

5.4 EXPERIMENTATION

For our experiments with tracking different constellations, we had two major setups: static and dynamic. The static tests involved having the Tango device and constellations stationary in the motion capture lab, described in Section 5.4.1. Section 5.4.2 describes our dynamic tests where we would move the constellation, the phone, or both. The data recording procedure was the same for each test, explained fully in Section 5.4.3. Section 5.4.4 details a node we used for measuring the positions of constellations before we actually collected our data. Sections 5.4.5 and 5.4.6 explain how we characterized the depth sensor on the Tango smart phone and our truth model. Finally, Section 5.4.7 describes the experiment that we conducted to estimate the maximum operating speed of our fiducial tracking system.

5.4.1 Static Experiment Setup

Our full static experiment requires many steps, listed below. More detailed explanations are offered in the referenced sections where applicable.

1. Arrange a test constellation of fiducial stickers on a poster board, see Section 5.3.3
2. Place the test constellation in the motion capture lab
3. Set up the Tango device in its holder on the tripod, see Section 5.3.2
4. Start a roscore on the data logging laptop

5. Track the Tango device and constellation(s) using the motion capture system
6. On the laptop, run the measuring tape node and record the pose, see Section 5.4.4
7. Turn off the motion capture system to remove extra IR light
8. Turn on the Tango data streamer application and IR light, see Section 5.6
9. Pick a Test ID string that very briefly and uniquely describes the test
10. Record all variables for the test in the variables spread sheet, see Section 5.4.3
11. On the laptop, record a few seconds of data data using rosbag, see Section 5.4.3
12. Name the bagged data with the Test ID

While each step contributes to validating the final calculations of our system, most components can be skipped when creating data for a specific quick test. An example of this is short tests where we only recorded a few images of very similar fiducial constellations purely for attempting constellation identification. The only data that needs recording is the images and knowledge of which constellation is within view. Takings steps like this helped us to save time as we implemented various parts of our final system. Figure 20 shows a basic test where we used only the Tango sensors to determine if there was enough ambient IR light to track fiducials.

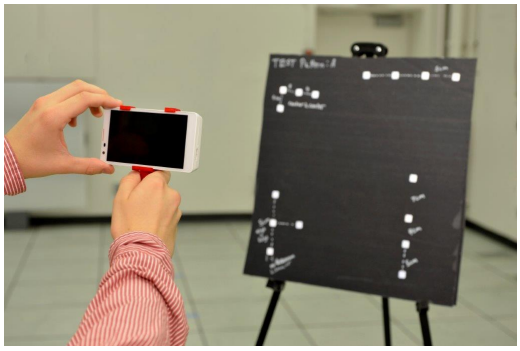


Figure 20. Basic experiment setup with just the Tango pointed at a poster board of constellations. Picture to be updated.

For the static experiments, we varied the distance and orientation of the target constellation with relation to the Tango device. The distances varied from 1 meter to 3 meters in increments of approximately 0.5 meters. We performed all of these tests with the IR filter over the RGB-IR camera, which did not allow us to put the test constellation anywhere other than the center of the cameras field of view due to the vignetting described in Section 7. At each distance we turned and tilted the constellation by approximately 0, 15 and 30 degrees. We used the motion capture lab to take these measurements, and rather than go for specific locations and orientations, we simply recorded the positions we did get with high precision. Table A.1 summarizes all the combinations of distance and orientation that we captured with the static tests.

5.4.2 Dynamic Experiment Setup

While it is very similar to the static experiment described in Section 5.4.1, the dynamic tests are actually much harder because the motion capture system has to stay on during the test. The motion capture system is great because we can get the 6 DoF poses of objects over time for validation purposes, but those cameras also emit a lot of extra IR light. This IR light illuminates additional fiducials or noise, but also directly interferes with our Tango device because they are strobing out of sync with the Tango cameras. Otherwise, the dynamic test is setup the same as static test, with the steps listed below.

1. Arrange a test constellation of fiducial stickers on a poster board, see Section 5.3.3
2. Set up the Tango device in its holder on the tripod, see Section 5.3.2
3. Start a roscore on the data logging laptop
4. Track the Tango device and constellation(s) using the motion capture system
5. Turn on the Tango data streamer application and IR light, see Section 5.6
6. Pick a Test ID string that very briefly and uniquely describes the test
7. Record all variables for the test in the variables spread sheet, see Section 5.4.3
8. On the laptop, record a few seconds of data data using rosbag, see Section 5.4.3
9. Move the constellation around while recording data
10. Name the bagged data with the Test ID

It is important to note that the movement of the constellation was done by one of us in motion capture lab. We decided to have a person do it instead of some mechanical system that would be perfectly consistent because we have extremely high quality pose verification from the motion capture lab and our final goal is to track a person or have a person walking and using it to track other objects. Therefore, the movement in the tests should involve a person moving in reasonable ways, and capturing a very wide range of motion, angles, and distances. In the future, it may turn out to be beneficial to place the constellations on a turtlebot and automate the movement part of the test and have a person hold the Tango phone as it tracks.

5.4.3 Data Recording

For our experiments we have a two step data recording process. The first step is picking a Test ID to identify the test and record that along with all testing variables in a spreadsheet. This spreadsheet is saved in comma separated variable (csv) format so that Python scripts in Section 5.5 can read them when conducting and reporting tests. The csv format was chosen because the input is human readable and output is also easy to work with in Python scripts. Some of the variables include: the date, lighting and noise conditions, distance of the poster board to the Tango

device, data streaming settings, constellations and their poses and the fiducials that make them up. Through the pose and constellation information, the 3D locations of the fiducials or expected 2D image coordinates can be calculated for the static tests, giving a set of truth data for validation. There is also a notes section for describing motion during a test or any possible problems.

The second step is launching rosbag to record the ROS messages on specific topics being streamed from the Tango device. Rosbag allows us to record and playback ROS messages with the same timing as they were originally published, along with looping and many more options. The main topic we recorded is $\backslash camera \backslash luminance \backslash image\_raw$, which contains the luminance image explained in Section 5.6. For dynamic tests we also recorded the poses of the constellations published on the $\backslash tf$, short for transform, topic. We picked only these topics to keep the rosbags small and only have what is necessary for testing our system. The launch file that has these settings automated is located in the mobilep_mocap_launch package and is titled *record_tango.launch*.

5.4.4 Measuring Tape Node

To assist with setting up static tests, we developed a ROS measuring tape node that would read in data from the motion capture system, then calculate and print the poses of constellations with relation to the Tango device. These values are recorded by hand into the csv file mentioned in Section 5.4.3. Figure 21 shows the transformations involved and that we can calculate the relative pose as described in Section 1.2. Because objects don't move during the static test, this node only has to be run while setting up a specific configuration, at which point the values can be copied into the variables spreadsheet described in Section 5.4.3.

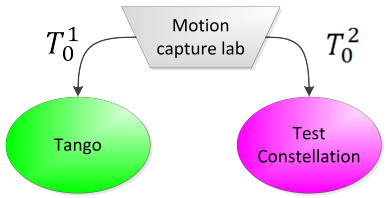


Figure 21. Measuring tape transformations given by the motion capture system during experimentation.

5.4.5 Depth Sensor Characterization

One of our original plans for implementing this mobile motion capture system was to utilize the depth sensor on the Tango device to get the 3D positions of fiducials and constellations. Before doing so, we prepared a small experiment to get a better idea for the quality of the depth readings. The experiment was to run the Tango data streaming application from Section 5.6 and have a new test node calculate the average depth value from the published depth images. We set up the our Tango smart phone holder from Section 5.3.2 on a tripod and pointed it at a flat wall. Using a level and measuring tape to measure the actual distance, we repeated the test a few times from 0.5 meters to 3 meters in half meter increments. See Section 6.5 for the results of this experiment.

5.4.6 Characterizing our Truth Model

In order to approximate the quality of measurements given by the motion capture system, which we generally used as truth data, we placed fiducial stickers on a poster board and measured their distances by hand and with the motion capture system. Because the motion capture software the provides the distances of each fiducial from the centroid of the grouping, we transformed the distances we measured by hand to also be with relation to the centroid, with the resulting measurements shown in Table A.2. These measurements are compared in Section 6.1.

5.4.7 Testing Operating Speed

In order to determine the operating speed of our system, we collected some data that would represent different usage scenarios and wrote a ROS node to publish that data at a fix speed. The different data scenarios are described in Table 10, with each row representing a different image. These different data sets provide varying noise scenarios, with the first case having noise without a constellation, the second two having constellations only, and the rest having constellations as well as extra points. The number of sub-constellations is provided because a great deal of our algorithms deal with finding and identifying lines as sub-constellations. While most of our system will mostly run in noiseless environments, we collected these bags to estimate the performance in the presence of minor noise.

TABLE 10

The various tests, by row, used to stress test our system operating speed. Each column indicates the contents of a test image in terms of fiducials and constellations.

Fiducials	Constellations	Sub-Constellations
6	0	0
12	1	4
16	1	4
20	1	5
22	1	5
22	1	5
26	2	5
32	2	8
38	2	9

The node we developed for this experiment utilizes the Python API provided for use with rosbag to pull an image out of a bag, which it then publishes at the requested speed [22]. We send the same image repeatedly to control the experiment better. These images are subscribed to by our fiducial tracking and pose estimation system as if it were live data. The tracking system was modified to attempt to find a constellation and estimate its pose for every image received and also

publish a message upon completion, whether successful or not. Also, all debugging messages and visualization features were disabled for the purposes of this test.

For the actual experiment, we ran the image publishing along side our pose tracking node. To double check the publishing rate of our node, we would use the ROS command `rostopic hz TOPIC_NAME`, which calculates an average rate that messages are being sent on a ROS topic as well as the standard deviation. The same command was used to determine the output rate of our tracking node. As we ran each test image, we tried multiple increasing publishing rates until the tracking system could not keep up with its input. These maximum operating speeds were recorded and are shown in Section 6.4.

5.5 AUTOMATED TESTING

In order to validate our implementation and ensure correct algorithm implementations, we utilized unittest, which is a Python framework for unit testing [25]. This framework allows for repeatable tests that run our code, check the outputs for specific values, and report failures. These failures include additional output that allows us to pinpoint parts of our implementations that need attention. By writing tests, we were able to confirm correctness and continue developing additional software and optimizations without breaking previously written code and features.

To perform system level tests, we utilized the rostest framework which is built on top of unittest [26]. This process is similar to the regular unit tests previously mentioned, but involves making ROS nodes out of those tests capable of subscribing to topics from live nodes and analyzing their output as they run. The test nodes run with nodes from our mobile motion capture system fueled by image and pose data from the data collected with rosbag, see Section 5.4.3. Typically, we wrote these nodes to wait for a given amount of time to collect a series of messages published across various topics, then examine the series of messages as a whole.

In addition to tests of correctness, we also used these tests to help with analytics. While system performance may pass tests by achieving a minimum error, we have the testing framework automatically log that error, and other calculations, so that we may examine the results between multiple tests and determine what variables are causing what irregularities may occur in our results. Truth data is read in from the csv file of test variables or from data recorded using rosbag. A Python shelve db is used to store the test results [27]. Every test run automatically updates its results in this database, which is also version controlled through the use of git, allowing us to retrieve specific data about any test at any point in time.

5.6 TANGO DATA APPLICATION

The Google Project Tango device was built running the Android mobile operating system. With that in mind, the easiest way to get the data we need off of the device was to create an Android Java application. The Tango has API's for accessing most of the data necessary. In order to stream that data in a way we can easily use, we used the ROS environment. ROSJava and ROSAndroid are both experimental ROS systems but are stable enough for our purposes.

OLogic, a robotics company in California, has recently been working on adding ROS support for to the Tango. Their git repository [28] has Android apps that get data off the device, and then publish the data as appropriate ROS messages. Specifically Ologic has extracted the depth and pose data. This gives us a great starting point, as this leaves us with finding the IR data and unifying this into a single application that publishes all the data at the same time.

5.6.1 Tango Superframe Access

All of the relevant image data taken from the Tango's cameras is stored in one large YUV formatted image known as the Superframe. This is also where calculated depth readings are stored. To get the specific images or data that you want, careful indexing and copying must be done.

Before the Superframe can be parsed, it has to first be retrieved. There are two methods for this that each have their advantages and disadvantages. The first method involves simply opening the device's default camera. On a normal Android device this would just open the back-facing camera. The Tango, of course, has many back-facing cameras, so by opening the default camera it actually is triggering all of the back-facing cameras to be open and combine them into this single Superframe. By setting the PreviewCallback of the opened camera to your own PreviewCallback class, every time the Superframe is updated it will make a call to that callback class to alert it of the update and pass the Superframe data.

The biggest benefit to opening the camera and setting up a custom PreviewCallback is that you have control over the regular android camera parameters. The settings of interest to us are the auto exposure settings, exposure compensation, and most importantly the Superframe "mode". This mode lets us choose what image to use for the large 4 Mega Pixel (MP) portion of the Superframe. Options are the RGB image or the raw IR image. The camera object also has a method for Previewing, which is a convenient and easy way to display the image on the tango device itself.

The problem with obtaining the Superframe this way is that, to our knowledge, we have no way of obtaining the pose information the Tango is calculating at the same time. Normally, to obtain the pose a VIOservice object, which is a Tango API feature that lets us query data from the Tango, is created. When attempting to include this with existing code involving opening the camera, the app would fail. With little documentation and the lower level source code being hidden it is hard to say exactly why this is the case. We have hypothesized that the VIOService is trying to get the data from the Tango the same way we are, and needs to gain access to the open camera streams. If we open them ourselves, the pose estimator can not.

Fortunately, the Superframe can also be obtained through the VIOService with a simple call to getCurrentSuperframe(). The downside here is that we lose the ability to set the camera parameters ourselves. This means that we can not tell the Superframe which mode to be in, RGB or IR. The Superframe from the Java pose API defaults to using the 4MP RGB image. Since the true IR cannot be accessed in this fashion, it means we have to make due with the RGB image which is only partially IR information. We explain our process of dealing with this problem in Section 5.6.4 and Section 5.7.1.

5.6.2 Tango Superframe Format

As mentioned previously, the Superframe is formatted as a single YUV image. Specifically this is YUV420SP NV21 format, which is the format used across all Android device camera images. A YUV image contains 3 planes of data. The Y plane contains the intensity of each pixel in the image, or the luminosity. The U and V planes are each a quarter the size of the Y plane, and together can be combined with the Y plane to produce the color R, G, and B values for each pixel.

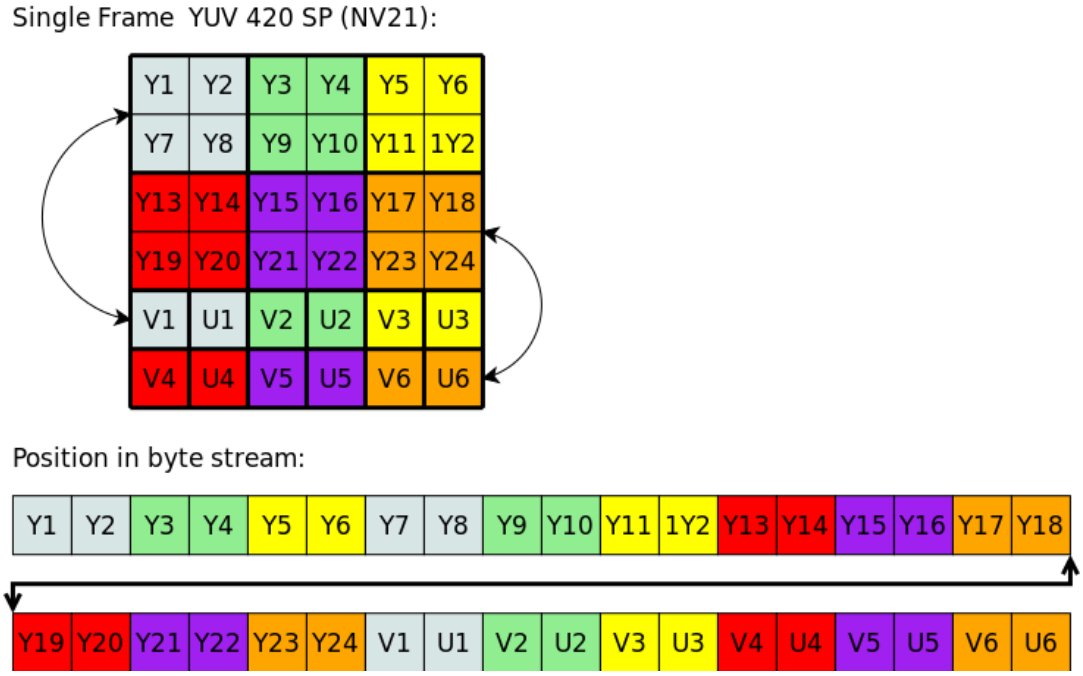


Figure 22. YUV420sp NV21 format reference, the image format used by Android camera images

As shown in Figure 22, the Y plane is contained in the first half of the image. In YUV420SP NV21, the U and V planes are in the second half of the image and are actually inter-weaved in VUVU ordering, so the first value from each plane come after the other, and then the second, etc. until the end of the image. The color coding in Figure 22 illustrates how the UV values pair with the Y values. As you can see the four top left pixels are colored gray, and the first two UV values are also gray. For every four pixel block in the Y plane there is an equivalent U and V value.

To convert a Y value intensity back to R, G, and B the following conversions can be done with it's corresponding U and V pair.

```
float r = y + 1.772 * v ;
float g = y - 0.344 * v + 0.714 * u;
float b = y + 1.402 * u;
```

The final step is simply to make sure that the values can fit into an 8 bit value. If any of the color values end up being above 255 or below 0, force them them to be 255 or 0.

Not every piece of data in the YUV formatted Superframe uses the UV plane, since not everything actually has color information. This is the case for the fish-eye image, the depth image, as well as random bits of header information. In these cases the UV plane data is filled with placeholder values and is ignored. Since the UV planes are preserved when not needed, it is possible to access desired pieces with or without color. With known offsets of the data given in a Superframe class, indexing into these pieces and pulling out what is needed is straightforward, and is discussed in the next three sections.

5.6.3 Depth Image Parsing

The depth image, as previously mentioned, does not contain any color data with it; all of the information we need is inside of the Y Plane. When the depth data is calculated by the Tango, it is stored as a 320 x 180 pixel image. Each pixel in this case is actually a two byte integer representing the distance away in millimeters that xy pixel position is away from the camera.

To extract the data, we begin by getting the starting index of the depth data in the Y plane which is stored as a constant in the Superframe class. We can then begin to iterate through the Y plane byte by byte. On every iteration we pull out the 2 next bytes from the Y plane and convert them to Integers which represent millimeters and save them in a new integer array. To do the conversion, the bytes must be combined to make one word by shifting the first byte to the left by 8 bits and performing a logical OR.

```
int msb = (int) data[YPlaneIndex + 1];
int lsb = (int) data[YPlaneIndex];
int pixDepthMm = ((msb << 8) & 0xff00) | (lsb & 0x00ff);
```

With an Integer array representing the depth image, we publish the data as an Image message in ROS.

5.6.4 4MP RGB-IR Image Parsing

In the Superframe, the last piece of data contained in the Y-Plane is the 4 MP image from the RGB-IR camera. Since color information is contained in this image, we have to pull data from the Y-Plane and the UV-Plane. Essentially, we want to construct a smaller YUV formatted image. This means slicing out the Y and UV plane separately and combining them back into one array. The Y-Plane data is between starting index defined in the Superframe class and the same value plus the size of the image, which in this case is 1280x720. Since the UV plane is half the size of the Y plane, the UV plane starting index can be calculated by calculating $EndY + 1 + OtherYSize/2$.

OtherYSize in this case is the size of the Y plane without the 4MP RGB-IR image we are trying to extract. The ending UV index is the starting UV index plus half the size of RGB-IR image.

In our case we are not very interested in the color data since we are only looking for intensity of IR light. This means we can pull out just the Y plane data to get a luminance only image. The luminance image can then be sent as is in a ROS Image message with the encoding of MONO8.

5.6.5 Tango Pose Access

The Tango's pose is very easy to retrieve when using the Java Tango API. Specifically, the API gives us a class called VinsServiceHelper. The function for getting the Pose information is getStateInFullStateFormat(). This means if we want to stream the pose data we need to constantly make a call for the pose in loop. For this to be safe and easy in Android, ROSJava has a class called CancellableLoop which allows it to be interrupted if something goes wrong. By overriding the loop() method we can call all the updates we need in order to stream the most up to date data. With the Pose data we are then able to publish this as both a transform in the TF Tree, as well as a ROS Pose message.

5.7 FIDUCIAL TRACKING

The Fiducial Tracker System must be able to take in a raw ROS Image stream from the Tango and be able to identify fiducial markers in the scene In order to do this there are four major steps that have to be addressed. Figure 23 outlines these steps. We must first be able to convert the black and white image from the Tango into a binary image in order to only focus on the brightest areas in the scene, which are the fiducial markers, described in Section 5.7.1. The binary image then has to be searched and group the white areas into clusters representing a possible marker, which is explained in depth in Section 5.7.2. Section 5.7.3 elaborates on how these pixels can then be averaged per cluster to obtain the centroid of each marker, then eliminated entirely if it is determined to be noise.

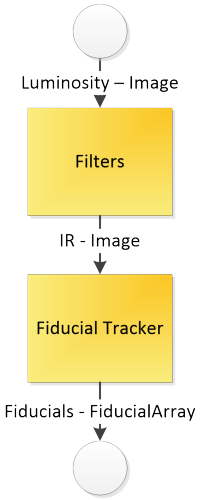


Figure 23. Fiducial tracking system overview.

5.7.1 Binary Image Filtering

A really simple way to produce a binary image from a mono gray-scale picture is to check each pixel against a threshold. OpenCV provides a method for this filter, which is appropriately named threshold(). This takes the image to filter, a threshold, the maximum value the image could have, and the mode to filter by. For our case we want a binary filter. This method works by simply checking if each pixel is above or below the given threshold. If a pixel is below the threshold, then it is assigned a value of 0, or black. If it is above the threshold then it is assigned a value of 1, or white. Through observation during testing, a good threshold for extracting only the reflective fiducials is about 75% of the brightest pixel in the image. This threshold can be increased when the entire scene is particularly bright which would lead to noise in the form of extra areas being passed through the filter. Increasing the threshold much more begins to make the fiducials smaller by cutting out the less bright edges.

Figure 24. Raw luminance image on the left, filtered binary image on the right with only fiducials present.

As shown in Figure 24, the threshold filter works fairly well when the scene is has low intensity light and the fiducials are much brighter than everything else. A lot of noise can enter the scene when the lights are more intense, most notably when they reflect off of white walls, creating large bright spots getting through the threshold filter. We implemented noise removal techniques in Section 5.7.3, but found this is not always good enough for large blotches and fuzzy edges. To remove any possibility of these bright spots a $850nm$ IR filter was purchased and put over the RGB-IR image to filter out all visible light, leaving only the IR light that is reflected off our fiducials and possibly other reflective surfaces as shown in Figure 25. This is much easier to deal with than the possible hundreds of extra points when no IR filter is used. Even though the image with IR filter is so noise free, the threshold filter still has to be run to convert the grayscale image to a binary image since our other algorithms depend on this structure.

5.7.2 Fiducial Marker Searching

Once we have a binary image, we can then search the image for all of the markers which would show as white in the binary image. To do this, a variation of a connected components search must be done. The goal of any connected components search is to search over the image for large groups of pixels that are connected together. If they are we, can save them in a cluster list to look at later. At the end of the algorithm we have a list of clusters.

Figure 25. Raw luminance image with no IR filter on the left, and raw luminance with an 850nm IR filter on the right

A common implementation of a connected components algorithm is to iterate over the binary image checking for a white pixel. When a white pixel is found, a breadth or depth first search is performed in a four or eight connected fashion from the pixel in order to find more white pixels connected to the initial pixel. Four or eight connected is a way of defining what is considered to be connected to a pixel, and references how many pixels are connected to any other pixel. In Figure 26 and Figure 27 the bright blue pixel in the middle is the current pixel, and the green pixels around are the pixels that are considered connected to it and will be searched. The inside search ceases when no further connected pixels are found, and the process continues to iterate through the rest of the pixels until it finds a new white pixel it has seen before and can again search for connected pixels. A psuedo code version of our implementation is described in Algorithm 1

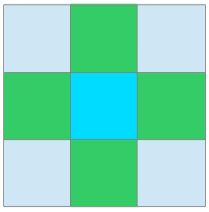


Figure 26. four connected search pattern

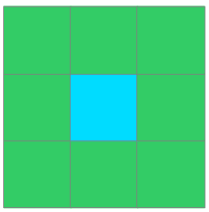


Figure 27. eight connected search pattern

Algorithm 1 Our connected-components implementation

```
1: clusters ← empty_dictionary
2: currentCluster ← 1
3: visitedLabels ← 2D_array_of_size_width_by_height
4: i, j ← 0
5: for all columns of the image do
6:     for all rows of the image do
7:         if pixel at position (row,column) is white and is not null in visitedLabels then
8:             stack ← initialized with (row, column)
9:             visitedLabels[row, column] ← marked as currentCluster
10:            while stack is not empty do
11:                currentPixel ← pop from stack
12:                neighbors ← get 4 connected pixels of currentPixel
13:                for all neighbors pixel positions (x,y) do
14:                    if (x,y) position is in range of image then
15:                        if pixel at position (x,y) is white and is not null in visitedLabels then
16:                            visitedLabels[x, y] ← marked as currentCluster
17:                            push (x,y) onto stack
18:                            append (x,y) to cluster[currentCluster]
19:                        end if
20:                    end if
21:                end for
22:            end while
23:            increment currentCluster
24:        end if
25:    end for
26: end for
```

We implemented this exact version ourselves, but it was decided that it was running too slow to be of much use in a real time system. Instead of spending time to rewrite and optimize, OpenCV again proved to be useful by having a slightly different version known as contour searching already implemented and optimized for speed. The method findCountours behaves similar to our implementation, but instead of searching for the contents of entire clusters it searches for just the contours or edges in the image. This algorithm implements the border following algorithm described in [29], which looks to classify the topological structure of the binary image by building a tree of contours where children are contours inside of others, or holes. [29] also provides an alternative which only searches for exterior contours.

5.7.3 Fiducial Centroid Calculation and Noise Removal

With a list of all the groups of pixels in the image, we are then able to calculate the position of each fiducial marker. This is done by a simple centroid calculation where the x and y values of each pixel in a cluster is averaged to sub-pixel precision for a more accurate centroid for the identification step in Section 5.9.3. Since the OpenCV findCountour function gives only the points on the edge of a marker, we have fewer values to use in the centroid calculation so the sub-pixel average is critical.

At this step, we can also take the time to remove noise from the system. If the scene has an abnormally bright region like a large bright light in the distance, then it will get through the binary filter as an obviously large portion that is not a real fiducial. In this case we can filter remove it from our list of clusters based solely on the amount of pixels inside of the returned contour list. If it is bigger than our largest observed fiducial then it is safe to throw it out.

The next step can be taken to filter out unwanted noise when considering the fact that the real fiducials will all be circular, and background noise groups will likely not be. By getting the perimeter and area of each cluster, we can determine its circularity using the metric below, as described in [30].

$$circularity = \frac{4 * \pi * area}{perimeter^2} \tag{4}$$

With this metric a value of 1.0 is a perfect circle, and anything else is considered less circular. When our fiducials are large enough in the picture, their circularity was found to be above .8 and trailed off to .75 the further away from the camera they got. Setting a circularity threshold to be .75 removes a large amount of unwanted detections in the scene, further reducing the amount of searching required, which is explained in Section 5.8

5.8 IMPLEMENTING RANSAC

Given the large number of fiducials found by the fiducial tracker in Section 5.7, we were in need of an efficient means of searching through those points for our predefined constellations. To accomplish this, we researched the RANSAC algorithm and implemented our own version. Sections 5.8.1 and 5.8.2 describes how RANSAC is usually implemented and how we implemented

it. We only used RANSAC to find groups of collinear points that could be linear constellations, then used those groups with the actual constellation identification process is described in Section 5.9.

5.8.1 RANSAC

A popular algorithm used for fitting a model to a data or finding a model in a set of data with noise is the random sample consensus algorithm, or RANSAC. Developed in 1980, RANSAC was designed to search a set of data and find a reasonably good subset of that data representing some model quickly by ignoring noisy outliers in the data. The model could be a line of best fit or point correspondences with another set of data. The basic idea is that the algorithm picks random seed points from a data set, approximates a model based on those seeds, then searches the rest of the data set for inliers that fit that model within some given *tolerance* of error. This process repeats itself for k iterations, where k is a predefined number, and the best model across those iterations is recorded. If a particular model has at least n inliers, the algorithm returns that model and exits early. The goal is to find a model in the given data that is good enough very quickly. Algorithm 2 shows the RANSAC algorithm in its most basic form.

Algorithm 2 The RANSAC algorithm.

```
1: Initialize best to hold the best match found so far
2: for k iterations do                                  ▷ k is some large number
3:     Pick seed points at random
4:     Approximate a model based on these points
5:     Intialize inliers
6:     for all other points do
7:         Calculate the error from the point to the model
8:         if error is less than tolerance  then        ▷ error is for tuning
9:             inliers ← point
10:        end if
11:    end for
12:    if  inliers has at least n inliers then          ▷ n is for tuning
13:        return The approximated model and inliers    ▷ Exiting early
14:    end if
15: end for
16: return best as the best match found or FAILURE if best sucks
```

The values for n and *tolerance* are application specific and describe how tightly fit a model must be to be considered good enough. While n must be less than or equal to the number of data points input, it can be very large if a very small amount of noise is expected. Similarly, the *tolerance* should be greater than zero unless the typical error per point is zero. An example described in [31] is searching for the best circle out of a set of 2D points. The process involves picking three points, which is the minimum number to define a circle, then finding other points whose distance from the center of that circle are within the given *tolerance*. A similar process can be used to find lines, other shapes or more complex models.

The value for k represents the minimum number of iterations that it should take to find a reasonable model when searching randomly. [31] provides the equation, $k = w^{-n}$, relating the number of iterations to the minimum number of inliers needed for a good model. The value w represents the probability that any given point will qualify as an inlier and must be estimated for a given application. Once k is calculated, [31] suggests adding two or three times the standard deviation of k to give a more reasonable estimate of how many iterations it will actually take.

5.8.2 Our Implementation of RANSAC

In order to find linear sub-constellations, we developed a version of RANSAC specific to our application. The goal was to find tightly fit lines in an image made of four points quickly and nearly 100% of the time. While considering some features of the original RANSAC algorithm described in Section 5.8.1, we decided on a few modifications while adhering to the spirit of RANSAC. The first is that we cache all lines that would normally be considered good enough to quit early because we want to find multiple lines in an image.

The next modification is that we pick seeds in a deterministic fashion instead of randomly, meaning we try to use every single combination of points as seeds. This guarantees that we will find all the lines in an image if they exist. Searching in this manner also sets the value of k to be $\binom{p}{2}$, where p is the number of points being searched at a time.

Our implementation of RANSAC is shown in Algorithm 3 below. The input is the set of all the 2D image coordinates of fiducials found in a single IR image. The output is groups of collinear points, where the groups are at least four points and those points are within some maximum distance from the line that best fits them.

The algorithm begins by creating a record of the results. Next it iterates through every combination of input fiducials, which will act as an exhaustive search. At each iteration, the two points are recorded as the inliers. Now the distance between the line connecting these points and every other point is calculated, explained in Section 5.8.3, and each point that is less than some tolerance away, measured in pixels, is added to the list of inliers. If the list of inliers has four or more fiducials, then it is a candidate for being a linear constellation, and gets added to the results list, which is returned at the end.

While writing this algorithm, we considered some optimizations to allow it to finish faster. One was to remove points from the data set when they are found to fit a model. We decided against this because points could be shared between lines in an image, either through noise or an intentionally created set of constellations. Similarly, we decided not to exit early when four lines are found because one of them could easily be the result of noise.

5.8.3 Calculating Distances to Inliers

In order to calculate the distance from a point to a line connecting to other points, we utilized some standard triangle math. We started by rewriting the standard area equation for a triangle as $h = 2 * A/b$. Here b is the base of the triangle or the distance between the two points making the line in question. We solve for the base using the distance equation. We are solving for h, which is the height of the triangle and also the distance of the third point from the line.

Algorithm 3 Our RANSAC implementation

```
results ← empty_list
for all pairs of found fiducials do
    inliers ← pair of points
    Estimate line connecting these points
    for all other points do
        calculate distance from the point to the line
        if distance is less than tolerance then
            inliers ← point
        end if
    end for
    if length of inliers is at least 4 then
        results ← inliers
    end if
end for
return results
```

This just leaves the area, A, which can be calculated as the one half the determinant of coordinate vertices mentioned previously. Equation (5) shows D_x and D_y which we represent the X and Y displacements of the two points forming the line in question, a and b. Equation (6) shows the final equation we used with the area of the triangle defined by the three points divided by distance between a and b equaling the distance of point c from the line connecting a and b.

$$D_x = x_b - x_a, D_y = y_b - y_a \tag{5}$$

$$distance = \frac{|D_y x_c - D_x y_c - x_a y_b + x_b y_a|}{\sqrt{(D_x)^2 + (D_y)^2}} \tag{6}$$

5.9 CONSTELLATION IDENTIFICATION

Section 5.7 described how all of the 2D fiducial locations are found, and Section 5.8 searched through all of these points to determine which sets of points form a line. With this information we can verify if the points in our scene are actually a constellation that we recognize and have measured. There are two steps that allow us to do that. The first, described in Section 5.9.3, is to identify if the lines we have found match any of the linear sub-constellations that are known. After sub-constellations have been identified in the scene, we can determine if these sub-constellations make up a complete constellation because we know which sub-constellations each constellation is made of.

5.9.1 Our Constellation Description

For us, a constellation is defined by any set of known configurations of lines. These known configurations of lines are referred to as sub-constellations. These sub-constellations are all uniquely identifiable, which means different sets of sub-constellations can together define a complete constellation. Figure 28 shows an example of a complete constellation, which is made up for 4 lines forming a rectangle. In this specific example the 4 lines are sharing points in the corners, but a constellation can be made up of any coplanar lines. Figure 29 shows a closer example of a line, or sub-constellation, of points that make up a constellation. The sub-constellations will be discussed in more detail in Section 5.9.3.

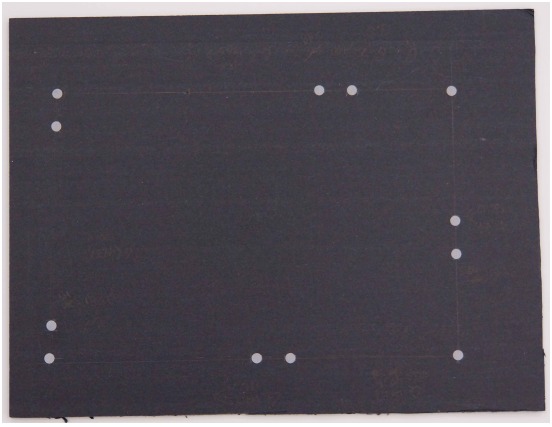


Figure 28. An example of one possible constellation configuration of points.

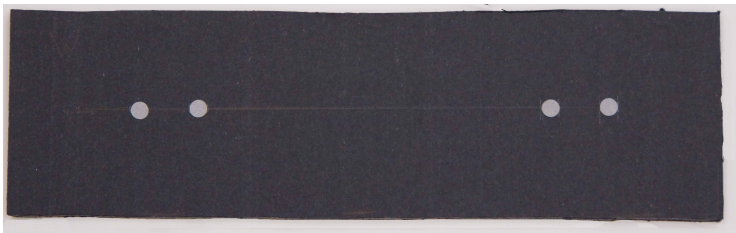


Figure 29. An example of one possible sub-constellation configuration of points.

5.9.2 Constellation Databases

In order to keep track of all the constellations and sub-constellations that could be found in the scene, a simple way to record and store them had to be established. A CSV file was determined to be the easiest input method for recording these constellations. Two separate CSV files, one for constellations and one for sub-constellations, are used for input into a database. A python script was made to read the CSV files and then load them into the database for easy access. The database we are using is called Shelve and is very simple [27]. It allows for us to keep a persistent

python dictionary by saving it to a file. We save the constellations and sub-constellations inside of this database once and then can access the data by interacting with it just like a normal python dictionary, taking away any complexities from other types of databases.

The sub-constellation CSV file is formatted like in Table 11,with sample data included. The top row has the labels of each column. The first column is the unique name for the sub constellation, and the 4 columns after that are the distances in mm from the first point along the same straight line.

TABLE 11

sub-constellation CSV format

Name	pt_A	pt_B	pt_C	pt_D
W	0	25	175	200
X	0	25	50	100
Y	0	50	75	150
Z	0	75	100	200

The sub-constellation CSV parser script ignores the first line, and then for every line after extracts the name and all 4 points. A fiducial array message object, defined in Section 5.2.4, is created for the line where each fiducial's x positon is the distance read in for each point, and the y and z values are left as 0. The fiducial array is inserted into the database, with the key being the name of the sub-constellation.

A constellation has different requirements in order to define it, so the constellation CSV is a little more complicated. It is shown in Table 12 with some sample data of a single constellation included. The first row contains the unique $constellationID$, with the first column being a label and the second being the constellation ID value itself. The second row tells exactly which sub-constellations together make this whole constellation. The third line is a row of labels which define the columns for the rows thereafter; the *name* field is the name of the sub-constellation and the x,y, and z columns identify the position of each of the points of the sub-constellation relative to the whole constellation's centroid. The rows after that are the actual point data for each sub-constellation. Multiple constellations can be put into the CSV by maintaining the same format, starting with the Constellation ID of the next constellation.

The parsed data from the CSV file is put into the database in a similar way to the sub-constellations. A constellation message object is created, defined in Section 5.2.3. Contained in this message is a sub-constellation array where each sub-constellation row is put into a sub-constellation message object which is defined in Section 5.2.2. The newly created constellation message is stored in the database with the constellation ID being used as the key.

TABLE 12

constellation CSV format

Const_ID	4			
Pts	L	M	N	O
Name	X_coord	Y_coord	Z_coord	
L	0	0.0993475	0	
	0	0	0	
	0	0.04928	0	
	0	0.14888	0	
	0	0.19923	0	
M	0.30071	0.08729	0	
	0.30071	0	0	
	0.30071	0.04959	0	
	0.30071	0.10034	0	
	0.30071	0.19923	0	
N	0.1365675	0	0	
	0	0	0	
	0.09431	0	0	
	0.15125	0	0	
	0.30071	0	0	
O	0.1508125	0.19923	0	
	0	0.19923	0	
	0.05104	0.19923	0	
	0.2515	0.19923	0	
	0.30071	0.19923	0	

5.9.3 Linear Sub-Constellation Identification

In order to identify and match linear constellations of points to pre-defined configurations, a mathematical concept known as the cross-ratio is exploited. The cross-ratio looks at different distances between the 4 points on that line. The distances used are annotated in Figure 30, where the points are A, B, C, and D, from left to right.

$$R(A, C; B, D) = (AC/BC)/(AD/DB) \tag{7}$$

The cross-ratio is a ratio of ratios, and it says that for any 4 collinear points A, B, C, and D then the ratio defined in Equation (7) is the same for any projection of this line in space. This means that when the sub-constellation is projected onto the image plane of our camera, the cross-ratio will be identical no matter the pose of the sub-constellation, including skews, as shown in Figure 31 [32].

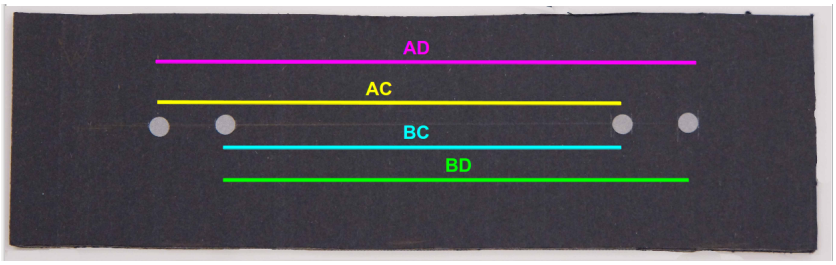


Figure 30. The 4 distances the cross ratio equation uses, annotated on a sub-constellation.

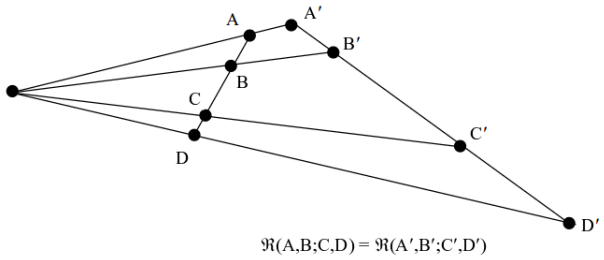


Figure 31. Different projections of the same line onto the image plane yield the same cross-ratio [32]

If you were to rearrange the ordering of the points used in the calculation, then there should be $4! = 24$ possible cross-ratios. The cross-ratio has a unique property that no matter the ordering of the 4 points on that line, the cross-ratio will simplify down to being 1 of 6 possible ratios due to duplication from mirroring. If the cross-ratio $R(A, B; C, D) = \lambda$, then the other possible 5 cross-ratios are $1 - \lambda$, $^1/_\lambda$, $\frac{1}{1-\lambda}$, $\frac{\lambda-1}{\lambda}$, $\frac{\lambda}{\lambda-1}$, as shown in [32].

For our purposes, we do not care about the other possible orderings. Since our lines are rigid and we are doing a non-random search over the image, we will find the points in the ordering $ABCD$, or $DCBA$ when the line is upside down. These orderings are mirrors of each other, which means that $R(A, B; C, D) = R(D, C; B, A)$ and we will only need to know one of the 6 cross-ratios.

Differently configured sub-constellations will result in a unique cross-ratio value for each sub-constellation. With the database of known sub-constellation configurations, defined in Section 5.9.2, the cross-ratio can be used as a very easy means of identifying which sub-constellations are currently in the image plane. Since there is error when using the found image points, the computed cross-ratio in image space will end up being slightly different from the truth cross-ratio calculated with real world measurements. To compensate for this, the percent difference between the image cross-ratio and the truth cross-ratio is computed, shown in Equation (8). If the percent difference is between some set threshold, we can classify the sub-constellation as one we know exists in our database.

$$\%difference = \left| \frac{R_{image} - R_{truth}}{R_{truth}} \right| \tag{8}$$

5.9.4 Constellation Identification

Knowing what sub-constellations appear in an image, we can simply match those sub-constellation to a bigger constellation in the database that has those sub-constellations. If a false positive was found for a given sub-constellation, it is also possible that a false positive could also be found for a regular constellation. In this case our system accepts it and treats it as a real identification. However, given our low noise environments, false positives are extremely unlikely. See Section 7.3.2 for ways to improve our system to help eliminate false positives.

5.10 CONSTELLATION POSE ESTIMATION

While estimating the 6 DoF poses of the constellations we tracked, we needed to take the image coordinates of our fiducials and estimate a pose in 3D space based on that information. We also had the real world measurements from the database mentioned in Section 5.9.2 and the RGB-IR camera's intrinsic parameters that describe the focal length, image optical center and distortion coefficients. With all of this information together, we approached our problem as the Perspective-n-Point (PnP) problem. PnP is often associated with pose estimation, but it is also used when calibrating cameras [33]. Specifically, the value for n here is 3, so we are working with the base case of PnP, P3P. This section outlines a general approach described by [33,34]. We chose the value 3 because the minimum number of points necesessary to describe a plane is 3 points, which is convenient because we are tracking planar constellations.

The first step is to prepare the found image coordinate vectors by projecting them onto the unit sphere using camera intrinsic parameters. This is done by subtracting the value for the optical image center then dividing by the focal length, all expressed as pixel values, in the x and y dimensions and setting the z dimension of the vector equal to 1. Next the vectors are normalized to be of unit length. This process will yield unit vectors pointing from the camera origin through each of the points in image space.

The next step is to build up a system of equations using the law of cosines. Equation (9) is a sample of that system of equations and is based on Figure 32. AB is a known distance stored in the database from Section 5.9.2. The result of the angle between image points u and v, $cos\alpha_{u,v}$, can be calculated because these points were projected into 3D vectors in the previous step. All that is left is PA and PB, which are the distances from the camera origin to the tracked fiducials in 3D space. The same process is used to build up the other equations to fully relate points A, B and C shown in Figure 33.

$$PA^2 + PB^2 2.PA.PB.cos\alpha_{u,v} = AB^2 \tag{9}$$

Efficient solutions to the system of equations are described in [33, 34]. The objective of this system of equations is to solve for the pose of a triangle made up by points A, B, and C by solving for the distances PA, PB, and PC shown in Figure 33. This process can yield up to four unique possible solutions for the three distances. Multiplying these distances by the unit vectors of the

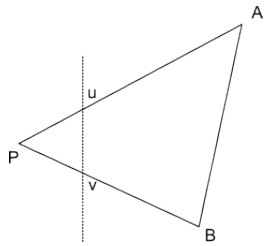


Figure 32. Diagram of found image coordinates and constellation points in 3D space related by Equation (9)*. This image was taken from [34].*

corresponding image coordinates will yield the 3D positions of the fiducials relative to the camera origin.

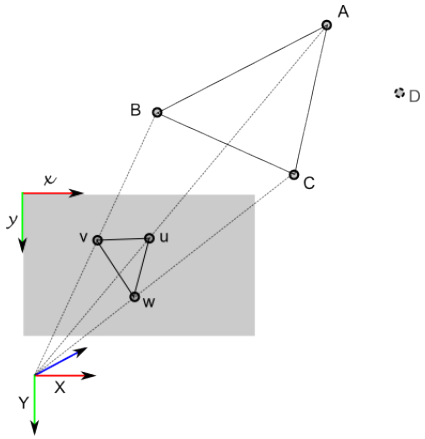


Figure 33. Diagram of the projections of image coordinates to 3D space used in the solution of P3P. Points A, B, C, and D are the centroids of sub-constellations. This image was taken from [34].

Next, each set of 3D positions of fiducials is used in a singular value decomposition to estimate the pose of the constellation, as described in [35, 36]. The result of this calculation is a translation and rotation from the camera origin into 3D space. Because multiple solutions are used here, multiple transformations have also been calculated and the best one is chosen by projecting the fourth point D using the z projection from each solution and the camera intrinsic parameters. The projection that fits the known geometries of the constellation best is the resulting pose of the found constellation.

PnP is a heavily studied problem in computer vision as well as many other fields. Knowing this, we searched for existing implementations of solutions and found the OpenCV function *solvePnP* at [37]. This function takes the known 3D relations of the points in a constellation, their current image coordinates and camera intrinsic parameters and returns the translation and rotation from camera origin to the centroid of the input points in 3D space. The use of this function saved a lot of development time and allowed us to fully prototype our system on time.

5.11 ROBOT FOLLOWING APPLICATION

For the purposes of demonstrating how our mobile motion capture system could be used on real robotics systems, we sought to mount the system on a mobile robot in order to have it follow a user around while staying a fixed distance in front of the user. This also demonstrates its use in augmented reality scenarios, as the user could be carrying a tracked tablet that can now be used as a proper AR display with spatial context. There are two main components that had to be developed for this system to work. Sections 5.11.1 to 5.11.3 discusses the pan and tilt turret for its use of orienting the Tango to always be pointed at the target. Section 5.11.4 explains the mobile robot driving behavior we implemented.

5.11.1 Dynamixel Servo Setup

The pan and tilt turret we chose to use was an off the shelf design discussed in Section 5.3, and uses dynamixel servos. Once the wiring is setup, you can connect the dynamixel servos to a computer through USB and command these servos to move to certain positions, in radians.

In order to get this working, we used the ROS package named *dynamixel_controllers* [38] which provides an easy way to get our turret up and running. Specifically, it provides a utility for bringing up each servo individually, and setting up server controller nodes that listen for control messages and actually handle talking with the servos. This lets us publish a float message to the topic *pan_controller/command* or *tilt_controller/command*, which directs the servo to move to the given angle in radians. Another nice feature about this package is that it is constantly publishing the status of each of the servos, with their current position, goal position, error, as well as operating temperature and if its currently moving.

While this package is a great utility for us, there was an extra step that we had to take in order to get both the pan and tilt servo working together. Out of the box, each servo is assigned an ID of 1, and in order to talk to multiple chained servos, each must have a unique ID. This involved using a script to manually connect each servo individually at a time and use the dynamixel driver command *set_id*, noting which id will later be used for pan and for tilt.

5.11.2 Robot Transform Tree Managing

With the addition of the pan-tilt turret and the turtlebot, there are many more transformations that have to be kept track of in order to complete the end goal of turtlebot following. In this system the Tango will be serving as the main sensor, providing a root global transform that everything else will be transformed under. Since the the Tango will be rotating independently from the turtlebot in order to keep track of the user, we also must note the current angles that the pan and tilt servos are at.

To accomplish this task the ROS Robot State Publisher utility was utilized. The Robot State Publisher keeps track of the current state of the robot by keeping its transform tree in check and constantly up to date. This takes the burden off of us to manually update all of the transform links we need in our model.

To model our robot, a Univeral Robot Description Format (URDF) file was constructed. A URDF file models the robot's many links and joints, and defines their relationship to one another. This model can be seen as a tree in Figure 34. With the model complete, the only thing we as a user of the Robot State Publisher have to do is publish the state of our pan and tilt joints, which are available from the package described in Section 5.11.1.

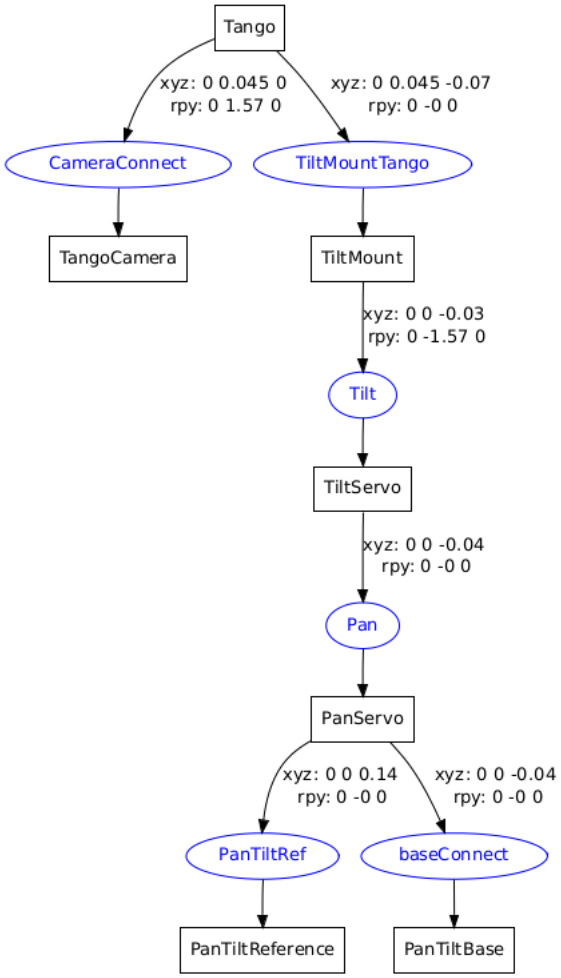


Figure 34. The transform tree of our robot.

In Figure 34, the boxes are physical links on our robot and that will have a transform published. The blue ovals represent joints on the robot, which may or may not move. The only joints that are not static are the pan and tilt joints, which represent angular displacement of the servo motors.

Important links to take note of in this tree are the Tango itself, the TangoCamera, and the PanTiltReference. The Tango is the root of our robot model, and its transform is not managed by

the Robot State Publisher. The position of the Tango link comes straight from the Tango itself, and the Robot State Publisher handles the publishing of all the links underneath the Tango root link. The TangoCamera is an important link because it defines the camera frame in which our objects are tracked in relation to. Camera frames use a different convention, with Z being forward. The Tango itself defines X being forward, so the TangoCamera link takes care of making sure the tracked objects will be able to be transformed under our Tango root link appropriately. The PanTiltReference is the frame that is used when we want to actually control the pan and tilt servos. It provides the zero position reference, where the turret and tango would be pointed straight ahead.

5.11.3 Pan and Tilt Tracking

With the transform tree being managed by the Robot State Publisher, which includes a pan-tilt reference frame, the task of controlling the servos to follow an object is very straight forward. The pose estimation part of our software, as described in Section 5.10, publishes a transform from our planar object to the camera frame itself. With the object now be published under the umbrella of our robot transform tree, we can now query for the transform from the object to our pan tilt reference frame. This gives us the ability to calculate the angular displacement of the the object in the pan and the tilt axis with relation to this reference frame. These angular displacements are the values that we could now publish directly to the pan and tilt servos. If the object were stationary, then the turret would move to correct the displacement with the goal being the object centered in the camera frame.

In order to perform this task live with the Tango, the speeds at which we were able to track had to be very low. The Tango adds weight to the turret making it more difficult to move, so fast movements would produce too much jitter and less stabile overall. In addition to having a less stable system, our ability to track successfully is significantly reduced due to having blurry images. Setting the speed of the pan servo to be at 0.1 radians per second and the tilt servo to be even less at 0.05 radians per second gave a fast enough response time to track objects at moderate human speeds, with enough stability to maintain these tracks.

5.11.4 Robot Driving Behavior

Currently, the turtlebot driving behavior is two basic rules to follow. The goal is to be 1.5 meters straight in front of the tracked constellation and pointed directly at the constellation. This behavior is accomplished with two rules and a set of tuning constants and error tolerances that guide that behavior. Since the turtlebot only has two drive wheels, the only commands we can send it are forward and angular velocity commands, meaning it can only rotate or drive forwards or backwards. However, both commands can be executed at the same time, resulting in an arc shaped trajectory. The following behaviors will be described in terms of driving forward or backwards and having a given angular velocity.

The simpler of the two rules handles the behavior of the robot when it is at or near its goal position. At this time, the only motion allowed is rotation that has the turtlebot point directly at the tracked constellation. Currently, this behavior triggers when the turtlebot is a quarter of a meter from its goal position. By having the turtlebot try to point straight at the constellation, it is preparing itself to either drive backwards or follow the constellation straight ahead very easily,

but makes moving side to side very difficult. Figure 35 shows the turtlebot rotating to point at the constellation once it is already near its goal position.

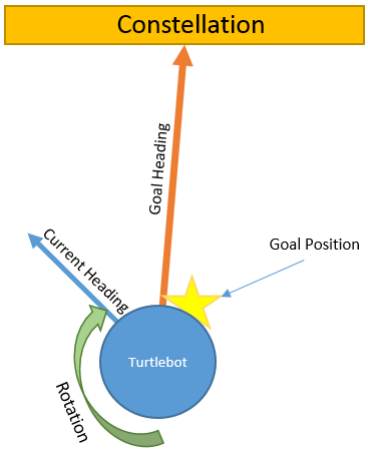


Figure 35. Illustration of the rotational driving behavior used for turtlebot following.

The second rule gives the driving behavior that causes the turtlebot to move towards the goal position after making a few decisions. First we calculate the error in the turtlebot's current heading, which is the angle between the current heading and the vector from the turtlebot to the goal. When the magnitude of this error is greater than 90 degrees, the goal is behind the turtlebot, which means the turtlebot should drive backwards to reach its goal. The next step is to calculate and send the actual commands for forward and angular velocity, which specify forward and backwards movement as well as turning clockwise or counterclockwise. We calculate command velocities by taking the minimum of a preset maximum velocity or the product of the error and a tuning constant. This allows the turtlebot to drive slower as it reaches a goal to prevent it from overshooting while still allowing a safe maximum velocity. By using this same process to calculate the angular velocity, the turtlebot will drive in an arc as it gets closer and closer to pointing at the goal position. Figure 36 illustrates the path the turtlebot might take as it approaches a goal position while moving forward.

As a safety feature, the tracking system notifies the driving node when it looses track of a constellation. As a result, a full stop command is issued, causing the robot to stop, and a beeping noise is also made from the turtlebot. The reason for these features is to prevent the turtlebot from driving blind and also to make it obvious to the users that the system has lost track. These features proved extremely useful while testing the full implementation of the system.

5.11.5 Robot Drive Simulation

In order to prototype driving behaviors in a safe environment, we utilized the ROS turtlesim package. This package models the same basic physics and movements of a turtlebot in a very easy to use environment. Figure 37 is a screen shot of how we used the simulator, showing two turtles

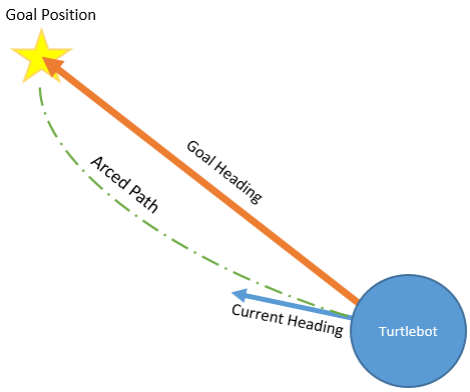


Figure 36. Illustration of the forward arc driving behavior used for turtlebot following.

and the paths they have taken around their environment. The simulation can be interacted with in the same way as a real turtlebot in terms of driving, and additionally provides the poses of all turtles currently in the simulation. By treating the green turtle as a tracked constellation and the other turtle as our mobile robot, we have created a simulated environment in which we can focus on our driving behavior without any dependence on our tracking system.

Figure 37. The ROS turtlesim program used to prototype turtlebot driving behavior.

The green turtle was manually controlled and the second turtlebot used our prototype driving code to attempt to get to a goal position directly in front of the manually controlled turtle. Because of the accuracy of this simulation, our driving algorithms were moved from the virtual world to the real world quickly and efficiently.

This page intentionally left blank.

6. RESULTS

In this section we present the general results of our mobile motion capture system as well as some basic observations made during implementation.

6.1 CHARACTERIZING OUR TRUTH MODEL

Table 13 shows the distance of each fiducial from the centroid of the group based on the measurements from table Table A.2. The average difference in measurement between the two methods is 0.177 millimeters with a standard deviation of 1.334 millimeters.

TABLE 13

Measurements made using the motion capture lab compared to measurements made by hand.

Fiducial Number	1	2	3	4	5	6
Manual Measurements	66.96	38.35	56.50	60.19	24.69	60.19
Motion Capture Measurements	69.33	37.29	57.02	58.99	24.40	60.92

Because these two methods of measurement are so close to each other, we decided to use the motion capture lab for all official measurements to compare to.

6.2 POSE ESTIMATION ERROR

After running through all the data collection from the experiments in Section 5.4, we calculated the average error in position and orientation for the pose estimates. Table 14 shows the average errors with respect to measurements made by the motion capture lab, as well as the range and standard deviation.

The orientations that we estimated are represented as quaternions, which are unit vectors that can be written as $a + bi + cj + dk$. Since quaternions can be tricky to compare, we used Equation (10) to get a representation of the angle between two quaternions. $< q1, q2 >$ denotes the inner product of two quaternions, which is $a_1 * a_2 + b_1 * b_2 + c_1 * c_2 + d_1 * d_2$. This method was specifically chosen because orientations in quaternion space are ambiguous, meaning there is more than one way to represent the same orientation. This equation accounts for that and provides a single angle from the known orientation to the estimated orientation.

$$\theta_{error} = cos^{-1}(2 * < q1, q2 >^2 - 1) \tag{10}$$

TABLE 14

Position error in each dimension relative to the camera as well as error in distance from the camera.

Error	Dimension (millimeters)			
	X	**Y**	**Z**	**Distance**
Average	-61.514	231.416	-28.449	-30.714
Range	0.722	0.670	13.654	13.657
Standard Deviation	0.233	0.153	3.424	3.446

After calculating the average error in orientation for all the tests in Section 5.4.1, we calculated an average error of 3.13 degrees. Additionally, the standard deviation is 0.041 degrees and the range is 0.19 degrees.

6.3 RESULTS OF THE CROSS RATIO

We wanted to determine how close the computed cross ratio is to what the expected cross ratio is from our hand measured truth model. To do this, we ran our system several times and recorded all of the error of every found sub-constellation against its known value in order to see what the average was. As shown in Figure 38, the average error was just under half a percent. This histogram also shows that the error never exceeds 5%, even though we defined the tolerance to be 20% for safety and maximize tracking probability. We believe errors that stretch far beyond the average are the result of hand measurement errors, as well as pixel error and jitter, which are both discussed in Section 7.

6.4 MAXIMUM OPERATING SPEED

Using the experiment described in Section 5.4.7, we determined the average operating speed when in the presence of different amounts of noise. Figure 39 shows the maximum output speed in the y-axis and number of fiducials in an image in the x-axis. For low noise scenarios, our system performed well above the reach goal of 30 Hz, marked by the red line. With medium amounts of noise the system still performed above the minimum goal of 10 Hz.

When each of the frequencies were recorded, the standard deviation was less than one-hundredth of a second. The complete results for this experiment are in Table A.3.

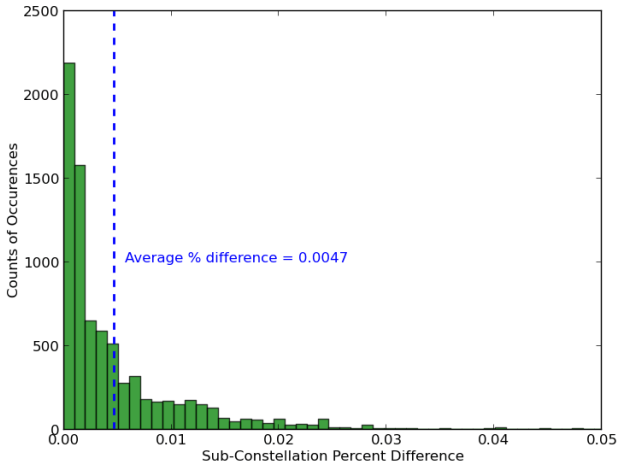


Figure 38. Histogram of the error of computed cross ratios of our sub constellation identifier.

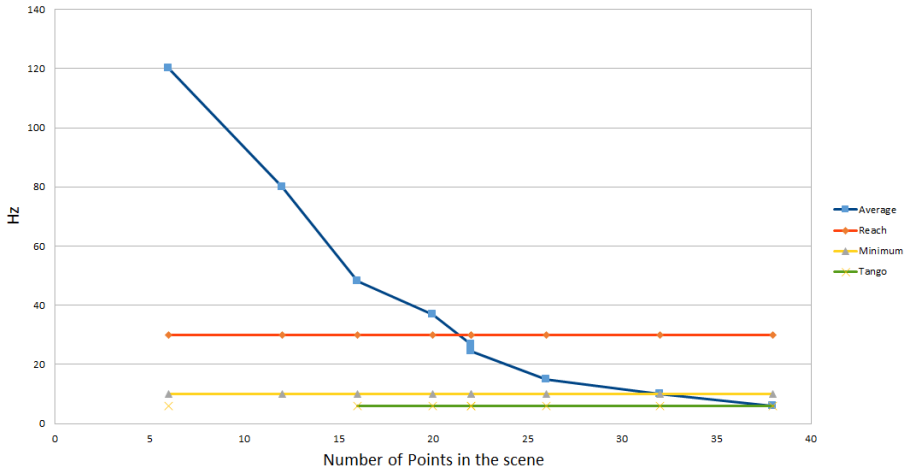


Figure 39. Results of operating speed stress tests of our mobile motion capture system against certain benchmarks.

6.5 DEPTH SENSOR CHARACTERIZATION RESULTS

After performing the experiment from Section 5.4.5, we decided not to use the Tango's depth sensor in our final implementation. At each distance, the average measurement was off by about 10 centimeters or more. During this experiment, we also discovered two common scenarios where holes appear in the depth image. Holes represent false readings where there should have been something. The first scenario, shown in Figure 40, is when IR reflective fiducials are in front of the depth sensor. Because these fiducials are so reflective, the IR light projected from the depth sensor bounces off and gives a poor reading. The second scenario is when an object is less than two thirds of a meter away from the depth sensor, causing large hole in the middle of the depth image. These holes are difficult to discern from actual readings and do not give the us data for the areas we are interested in. The rest of the time, the depth image didn't come out exactly flat as it should have when pointed at a flat wall.

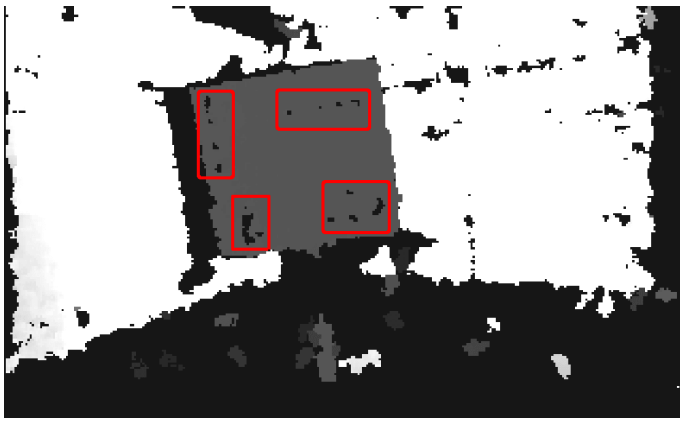


Figure 40. Tango depth sensor with reflective fiducials showing up as holes squared in red.

Other problems with the depth readings result from the Tango API only allowing us to get them at 5 Hz and at a resolution of 320 by 180 pixels. This is far too slow to work with our minimum system requirements. Additionally, this resolution is one sixteenth the resolution of the IR image, meaning we would lose a lot of useful data when upgrading the IR image to 3D space using this depth data.

7. DISCUSSION

This discussion will be driven by our system diagram, shown below in Figure 41. First we will discuss problems and limitations we encountered with the Tango device and the software that pushes data to the ROS environment. Moving on to our system implementation we will discuss how well each stage of our system performed. In addition to this, we will explain any sources of error that may have arisen in these stages as well as note anything we think could be improved in the future.

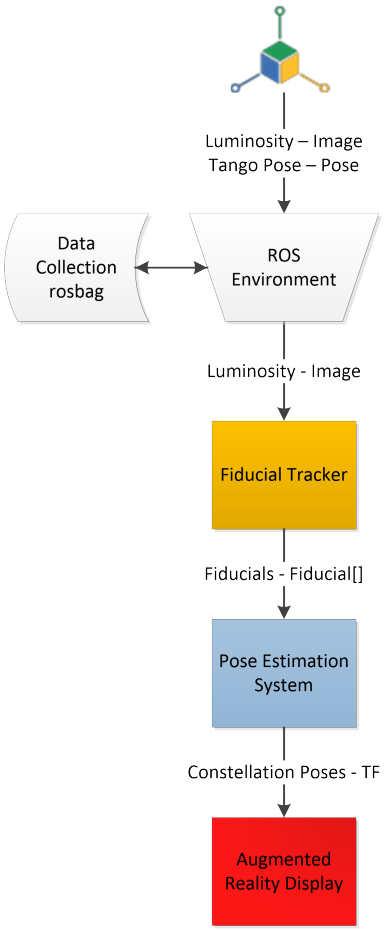


Figure 41. System overview of the major components in the Mobile Motion Capture pipeline.

7.1 PROJECT TANGO PROBLEMS AND DISCUSSION

In this section we discuss the problems and concerns we encountered when using the Google Project Tango device, and how certain parts of it were limiting our system. As a disclaimer, the

device we used was a prototype development kit from the Tango project, and, as of the time of this writing, the publicly available Tango device has not been released.

7.1.1 Device Limitations

The Google Project Tango Device is a very impressive piece of hardware and was the main driving force for our project, by lending us the ability to have a global reference frame from which we can track objects. The Tango was also one the largest limiting factor of our system. Being only it's infant stages, there are a lot of inherent problems with the device that we discovered. The device would typically last no longer than 10 continuous minutes while running our system before crashing or overheating, assuming it had not been used recently. This is obviously not practical when it comes to actually using our system in a real application.

In addition to the overheating problem, the Tango is also a limiting factor in the operating frequency of our entire system. While we have shown that our system can run quickly, we are still limited by the 8 Hz frequency that the Tango can output images. This severely limits the motion that we can capture during operation, and means that only slow movements can be accurately tracked.

7.1.2 Infrared Camera Problems and Concerns

While it was not a problem in the end, we originally had concerns with the RGB-IR camera on the Tango, which we planned to use as our main data source. This camera can see both visible and IR light, and displays both sets of data in one image. This was cause for concern because it meant that it might be possible that the IR Projector on the Tango could be seen in the RGB image. This is fortunately not the case. The IR Projector is intentionally operating out of sync with the RGB image so that the depth sensor can work independently of the RGB image functionality. Even though they are using the same sensor, the IR image is loaded at a different point in time in the duty cycle from the RGB image, so we don't see any of the IR projections.

Although the IR projector does not interfere, the RGB-IR image stream we were using is not ideal since there is visible light that we do not care about; anything in the visible light spectrum is noise in our system. To combat this problem we purchased an infrared filter to be placed over the lens to only let infrared light through. This had the unfortunate side effect of causing a vignette in our images, which severely limits our field of view. While still usable, the range was limited to the middle areas of the image. For future work, we recommend that a smaller and case-less IR filter be purchased. This would allow professional installation of the IR filter at a point which is closer to the image sensor itself, and would get rid of the vignette. Ideally, a custom filter could be cut to fit on the Tango as close to the image sensor as possible.

7.2 FIDUCIAL TRACKING PERFORMANCE

The fiducial tracking software is the first main component of our system, and is in charge of finding the fiducial markers in our input images in order to search and make sense of them later in the system. In this section we discuss this component's large issue with noise, examining a source of error in this component, as well as looking into alternative fiducials.

7.2.1 Dealing with Noise

Where the fiducial tracking software would encounter the most trouble was when there is a lot of noise in the scene, which was usually other bright reflective surfaces. This was especially a problem before using the IR filter. Anything overly bright would show up as overwhelmingly large noise sources and it would be impossible to only extract the fiducial markers. In attempts to deal with the noise, several filters were added like circularity metrics and maximum size metrics. These helped but ultimately proved the IR filter was necessary, which generally resulted in low to zero noise with the help of our other filters.

7.2.2 Alternate Fiducials

At one point we considered using larger square fiducials instead of the circular ones. The square fiducials are inherently larger then the circles, which means the squares can be tracked at a larger range than the smaller circles, and possibly be more accurate due to having more pixels to average with. While this is a great benefit to using the squares over the circles, they had two properties that made them less desirable. The first most obvious problem was that our system would likely classify them as noise with the circularity metric. The circularity metric is very effective tool in eliminating noise, and thus we decided to keep this metric and use circular fiducial stickers.

Another problem is actually the larger size of the square stickers, which implies the constellations must be bigger to adequately space the fiducials out. This is a concern because fiducials have to be a minimum distance apart to prevent our fiducial tracking system from treating them as a single blob. With the smaller circular constellations, we can create more compact constellations or more unique constellations that are the same size as the smallest constellation using large fiducials.

Due to time constraints, we were unable to use different kinds of fiducials outside of the square and circular fiducials. In the future, sphere shaped fiducials,like the one in Figure 4, would be ideal for attempting to track non-planar rigid bodies because the spheres can be seen from more vantage points than a typical planar fiducial by virtue of being three dimensional.

Another fiducial that we considered was IR LEDs. The advantage of having active fiducials is that an IR illuminator would be unnecessary because an IR camera would be able to see the fiducials despite the lack of ambient IR light. We did not use these because of time constraints and additional complexity involved with wiring LEDs to specifically function as fiducials. However, the software we implemented would not need any changes if the fiducials were changed, as long as they still show up brightly and circular in an IR image. Also, it may be advantageous to consider integrating IR LEDs when designing a custom electronic device intended for tracking, such as a new robot or custom augmented reality display.

7.3 CONSTELLATION DISCUSSIONS

Here we discuss various aspects of our the constellation identification system that could be improved, as well as some new features that could be introduced. Also included here is a discussion on the operating speed of our system because we found that our maximum speed was highly dependent on the identification algorithms.

7.3.1 Noise Classification and Operating Speed

To get an idea for how much noise we should expect our system to be able to handle, we walked around our lab with the Tango device and IR illuminator on and counted every thing showed up in the IR images. The only normal objects that showed up were particularly shiny whiteboards and TVs, which each showed up as a single large dot, as well as some backpacks and traffic cones that happened to have some reflective material. Because we had a motion capture lab, we also saw the original cameras as well as some random reflective fiducials on various tracked objects from other lab experiments. With all of these sources of noise, the only time our fiducial tracker registered more than 5 fiducials was when looking at a particular angle in the motion capture lab to see multiple cameras or when looking at constellations from past experiments.

With these objects in mind, we defined low noise scenarios as having 5 or fewer objects that could show up as fiducials in an IR image. In general, we found that we operated with zero noise. The experiment in Section 5.4.7 was conducted with images representing low noise scenarios as well as scenarios with even more noise to characterize expected operating speed under normal and more extreme conditions. Because of the sharp drop in operating speed with increased noise shown in Figure 39, we added a few basic checks to our algorithms that would allow them to give up early if too much noise is detected. One check added is to the RANSAC algorithm, causing it to exit early when too many potential sub-constellations are found. This has the effect of trading the probability of finding constellations for maintaining a fast operating speed when the noise increases unexpectedly. Modifying our constellation identification processes like this allows our system to operate at relatively the same speed even when there is an extreme amount of noise, although it is unlikely that any constellations will be found.

7.3.2 Eliminating False Positives

The current implementation of our system is susceptible to false positives when identifying constellations, as described in Section 5.9.4. False positives occur when things like IR lights and shiny objects show up bright enough to be considered fiducials and in rare circumstances form a line. Simply working in a place without these objects reduces the noise that our system encounters to next to nothing, but this isn't always an option. The following are two software improvements to reduce the number of false positives that we considered but did not implement.

The first option involves double checking values given from the PnP calculations. Using the transform and camera intrinsic parameters, the 3D locations of a found constellation can be transformed back into image coordinates and compared against the image coordinates of the found constellation in the current frame. If any of the sub-constellations are a false positive, the two sets of image coordinates will be drastically different and the found constellation should be considered a false positive and ignored.

Another option is to keep track of found constellation locations in image space and 3D space over time in an attempt to predict where a constellation will appear in the next image. An extended Kalman filter may be used for this, but searching a subset of a given image near a previously found constellation might be more efficient. This can also help to optimize tracking a constellation once it is found a first time. Here, false positives are detected when a constellation moves too far in a given

time step. This will not scale to tracking objects that can accelerate quickly and unpredictably and will not help at all with objects that are frequently lost.

7.3.3 Tracking Multiple Constellations

Due to time constraints and the low priority of this feature, we decided not to support the functionality to track multiple constellations at the same time. Also, using two constellations with 12 fiducials each would reduce our operating speed to about 20 Hz according to the results in Section 6.4.

Currently, if multiple constellations are present in a scene, the first constellation to be identified from the database will be the constellation whose pose is estimated. There is no way to predict which constellation will be identified first in the current implementation.

To make this system track more constellations, the pose estimation algorithm will have to be modified to not exit as soon as it finds a full constellation and then the pose estimation system will have to be modified to estimate multiple poses instead of just one. To make this work well, the underlying algorithms should be upgraded to get the operating speed back up to 30 Hz.

7.3.4 Cross Shaped Constellations

After implementing our current constellation model made up of four linear sub-constellations, we considered other constellation types but only one seemed feasible; cross shape constellations. These patterns would be made up of two perpendicular linear sub-constellations with a single shared point. Figure 42 is an example of this. The identification process will include finding the two linear sub-constellations and ensuring that the correct point is shared between them, requiring modifications to both the constellation database and constellation identification systems to indicate these properties. Additionally, the individual fiducial locations would have to be used when estimating the pose, as described in Section 5.10.

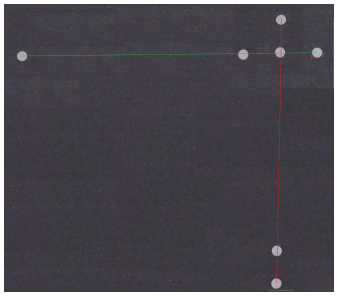


Figure 42. An alternate constellation pattern using two sub-constellations in a cross shape.

If properly implemented, the number of fiducials per constellation would be reduced from 12 to 7 fiducial stickers and the number of sub-constellations would be halved. These properties will allow for simpler constellations and a drastic decrease in processing time per frame. This design

was not implemented because it was too late in the development process after the entire system was implemented with the rectangular constellations.

7.4 DISCUSSING POSE ESTIMATION AND THE END RESULT

Here we discuss the results of the pose estimation system. Included are multiple solutions that can improve the performance of this system.

7.4.1 Pose Measurement Error

Based on the results of the error calculations done in Section 6.2, we did not meet all of our pose estimate goals in Section 4.1. The error calculated in position estimation is greater than 1.00 cm, which is our minimum goal in all dimensions. The standard deviation of these errors was far less than a centimeter in all dimensions. This can be visualized as a 1 cm cube that we estimate the centroid of tracked objects to be in. This cube is somewhere out in front of the Tango device, and stays the same size independent of how far it is from the device. On the other hand, the orientation was 3.13 degrees, which is better than the minimum goal of 5 degrees and very close to the reach goal of 2.5 degrees of error. This means that at any given time, the direction we estimate the object to be pointing in is about 3 degrees off from reality. This is excellent because three degrees is generally hard for humans to notice and is within our goals. The standard deviation for the orientation error was also very low.

Because the standard deviation for all of the position errors is so low, we can assume that there is a constant offset causing the position error. The simplest solution would be to subtract these errors values from the poses that we estimate. We have not yet done so because the error itself is so small and could be caused by multiple different things that we may or may not be able to control.

The most likely source of this error is that what we consider to be the centroid of a given constellation may be slightly different from what the motion capture lab decides is the centroid of the same constellation. This can easily happen for the motion capture lab when using fiducial stickers because that software tries to treat the stickers as spheres, which can move them from where they are in reality. If this difference is a few centimeters, the error in our system compared to the motion capture lab will also be a few centimeters and will be constant. With this assumption, the error can be tuned out of the system by measuring constellations more precisely and making sure they are defined identically in both our system and the motion capture lab.

Another way that a constant error could be introduced is through the camera intrinsic parameters explained in Section 1.5.6. Since these parameters are the values used to relate the image space of the camera to 3D space, all of our pose estimates depend on them. This means that if any of the intrinsic parameters are incorrect to any degree, the poses we estimate will be off by a constant amount. Theoretically, these intrinsic parameters could be tuned to a higher accuracy, resulting in better pose estimates. In practice, however, this is a very difficult task and it is often better to just be aware of the error in the system.

Finally, error could have been introduced by something as simple as human error. For example, we measured the constellations by hand before entering them into the database of constellations. If the reference constellation was off by any small amount, every pose estimate of the found constellation would also be incorrect by some constant error.

7.4.2 Homography

While considering using solutions to perspective-n-point implemented by OpenCV, we also considered an OpenCV implementation of homography to estimate the 6 DoF pose. Homography is a similar process to perspective-n-point, but utilizes camera projection principles to calculate a single solution rather than solving a complex system of equations. In terms of input, the homography function takes the same points as input, but transformed to image coordinates first. The homography function returns a homography matrix, which is a combination of translation, rotation and other various values. While complicated, the translation and rotation can be extracted from a homography matrix. Because of the increased complexity and the fact homography will only work with planar objects, we decided to continue to use the implementation described in Section 5.10 both to save development time and provide extensibility to 3D objects. However, the OpenCV homography functions could be used to better estimate the poses of planar constellations.

7.4.3 Improve Tracking Robustness

The system pipeline we implemented is currently linear; the input images produce an output of a 6DoF estimate of the pose of a tracked object. To improve the overall robustness of the tracking and final pose estimate, a feedback loop could be added. This would involve piping the output of the pose estimation system back into itself along with the input images. With this extra data, an Extended Kalman Filter could be used for correlating previous estimates to the current estimate. This would have the effect of smoothing out noise or bad readings, but would provide additional overhead in the software. Similarly, the fiducial tracking system could be optimized to search a subsection of an image where the fiducials are expected to be based on previous images.

7.4.4 Overall Resulting System

Overall, the mobile motion capture system that we implemented meets all of the other minimum goals as well as some of the reach goals. For example, we can track a single constellation anywhere from 0.5 meters to 3.5 meters away at any angle as long as the line of sight from the camera to the fiducials is not obstructed. We can also track constellations for 100% of the frames even while the constellation and Tango are both in motion, including when the Tango is mounted on a mobile robot. However, this motion has to be relatively slow and the tracking success is slightly dependent on how well we tuned the system. For example, a particular set of tuning parameters may be perfect for a space inside the lab, but the lighting in the hallways might be so different that the fiducials cannot be seen. While this is generally not the case due to the IR filter over the Tango's camera, there are lighting conditions that need to be tuned for.

7.5 TURRET CONTROL

The turret following code ended up working very well, but it had some minor limitations. Because of the weight of the Tango and IR i that went on top of the turret, moving the pan and tilt servos quickly became impossible without a large amount of jitter and instability. This jitter would end up bluring the images meaning tracking on the object would ultimately be lost. If we were able to run the servos at a higher speed and have them still be stable, it means that the response time would be much higher and more realistic human movements could be tracked.

This slow speed also affected the overall behavior of the driving. Since the turret would move slowly, the Turtlebot might end up rotating too much and the turret would lose track. This is partially due to time to tune the driving, but if the turret could move faster reliably this would be less of an issue, and the whole system could drive faster.

With this limitation in mind, we recommend that larger servos be purchased that can handle the weight of the Tango device better so that it can move faster with more stability.

7.6 TURTLEBOT DRIVING BEHAVIOR

While the implemented driving behavior used for following worked, it did have some flaws. The first is that it is running at only 2 Hz and all velocity commands are at a minimum in order to compensate for when our tracking software loses track when the Tango moves too suddenly. With a faster camera, we would be able to speed up the control loop for the driving and get smoother control.

Additionally, we did not fully tune this system because it was implemented outside of MITLL, where the driving conditions would definitely be different from inside the lab. Given enough time, the turtlebot driving code can have its speeds and error tolerances tuned to any driving surface, but those values will be different when moved to a new environment.

A future addition to the driving code would be speed control or incremental speed changes. This would limit the accelerations on the system and should allow the tracking system to keep up better because the images from the Tango should be less blurry. The driving itself will also be more elegant.

8. CONCLUSION

The goal of this Major Qualifying Project was to make a motion capture system that was not limited to a confined space like typical motion capture systems that are currently in use. The ability to track objects with high precision is a difficult task in robotics, and these motion capture systems provide a great way of solving this localization problem. Unfortunately current systems are not practical for robotics outside of a single room with tracking equipment.

Our mobile solution was all made possible by the advanced technologies present inside of the Google Project Tango, which features visual odometry pose estimation of the device itself, as well as an infra-red camera. When pairing this device with an IR illuminator, the Tango can see the same exact type of markers typical motion capture systems use.

The system that we implemented was the tracking software that would consume the sensor data provided by the Tango. The types of objects our system is able to track are planar objects, with unique configurations of these reflective fiducials. Image processing and computer vision techniques allowed us to develop a system that filters out only the reflective fiducials from the IR images, fit lines to these points in the image, and then analyze the lines to see if they are configurations we recognize. From there, the pose of the plane can actually be estimated by using prior knowledge of the positions of points on the plane.

Once we had a system capable of tracking these objects, we wanted to demonstrate how a robotics application could use our system. By mounting the Tango and illuminator on a pan and tilt turret, which was attached to a mobile robot, we wanted to have the mobile robot always keep track of the object by following it around.

After undergoing testing, it was determined that in our mobile motion capture system there was about 3cm in positional accuracy, with the object being tracked 90% of the time. The target plane could be tracked up to +/- 35 degrees in skew, and identified at any roll rotation. It could also track our object in the range of 0.5 - 3.5 meters. The entire system runs at about 10 Hz live, as this is the rate the Tango can output images; Our tracking software can actually run closer to 80Hz. With these results we believe the system we developed is a great success, capable of being a great tool for tracking and localization. This is further proven by the mobile robotic application that we implemented that uses our tracking system.

This page intentionally left blank.

APPENDIX A: THE FIRST APPENDIX

TABLE A.1

Details static test experiments describing the various poses used during the test. The tests are highlighted in groups of 5 to show orientations at each of the distances from 1 meter to 5 meters in increments of 0.5 meters.

Test Name	Position (meters)			Orientation (degrees)		
	X	Y	Z	Roll	Pitch	Yaw
PoseTest0	-0.03	0.03	1.08	2	5	90
PoseTest1	-0.01	0.06	1.09	14	4	93
PoseTest2	0.01	0.07	1.09	41	0	89
PoseTest3	-0.02	0.07	1.08	4	19	89
PoseTest4	-0.03	0.06	1.1	4	33	90
PoseTest5	0.11	-0.03	1.50	1	4	90
PoseTest6	-0.01	0.06	1.52	16	10	94
PoseTest7	0.14	-0.01	1.49	29	8	86
PoseTest8	-0.02	0.06	1.51	7	15	92
PoseTest9	-0.02	-0.01	1.52	10	30	94
PoseTest10	-0.08	0.02	2	1	4	91
PoseTest11	-0.08	0.08	2.01	9	8	91
PoseTest12	-0.07	0.1	2.02	27	0	92
PoseTest13	-0.09	0.17	2	4	18	89
PoseTest14	-0.09	0.15	2.02	4	30	90
PoseTest15	0.04	0.22	2.49	2	4	92
PoseTest16	0.04	0.24	2.48	14	6	92
PoseTest17	0.07	0.22	2.49	30	12	88
PoseTest18	0.05	0.2	2.5	0	19	92
PoseTest19	0.05	0.21	2.49	1	31	90
PoseTest20	0.05	0.21	3.01	1	6	90
PoseTest21	0.23	0.18	2.99	15	1	93
PoseTest22	0.24	0.17	2.99	22	19	87
PoseTest23	0.21	0.16	3	0	20	89
PoseTest24	0.22	0.16	3.01	0	32	90

TABLE A.2

Measurements of the same group of fiducials using two methods: by hand and using the motion capture system

	Hand Measurements			Motion Capture Measurements		
	X (m)	Y (m)	Z (m)	X (m)	Y (m)	Z (m)
1	55.06	-38.12	0.00	54.64	-42.68	0.04
2	4.24	38.12	0.00	5.80	36.83	-0.80
3	55.06	12.71	0.00	55.98	10.74	-1.44
4	-46.59	38.12	0.00	-43.97	39.32	0.05
5	-21.18	-12.71	0.00	-22.38	-9.60	-1.45
6	-46.59	-38.12	0.00	-50.00	-34.62	3.60

TABLE A.3

The results of stress testing our system.

Fiducials	**Max Hz**
6	120
12	80
16	48
20	36.8
22	26.7
22	24.5
26	15
32	10
38	6

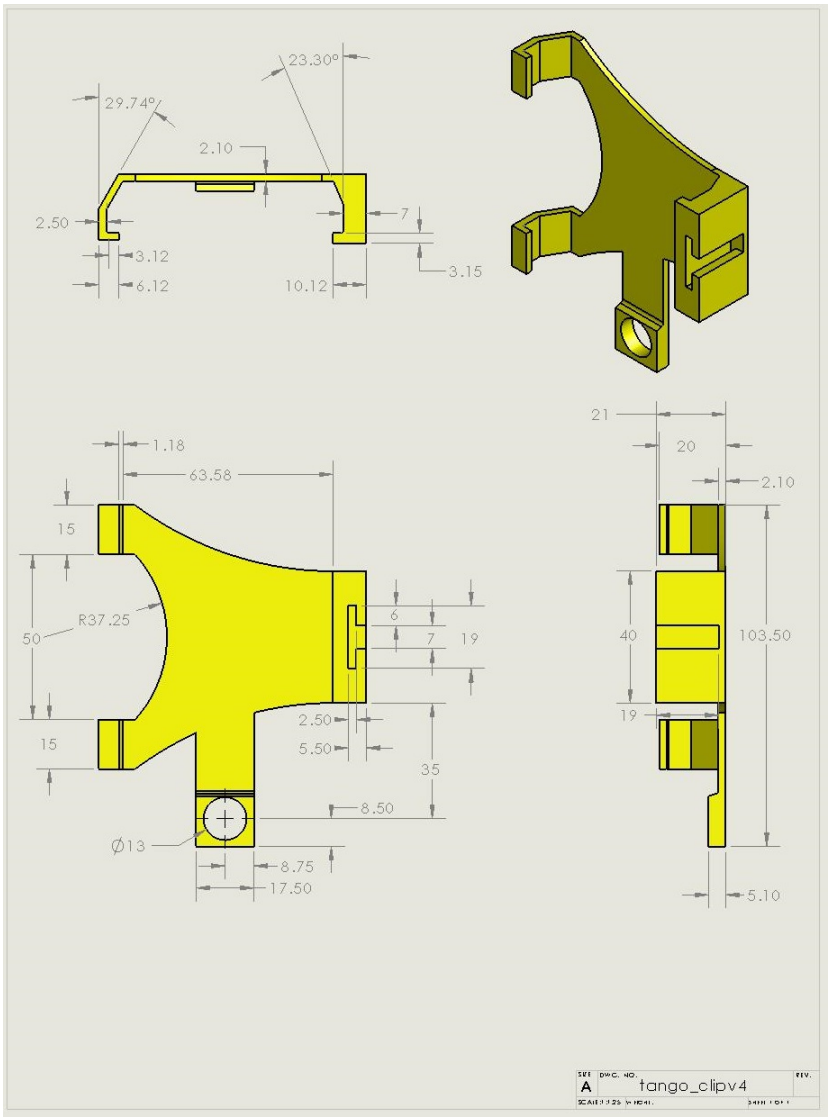


Figure A.1. Drawing file for the 3D printed Tango holder.

GLOSSARY

AR	Augmented Reality
DoF	Degree of Freedom
HRI	Human Robot Interaction
IR	Infrared
LL	Lincoln Laboratory
MIT	Massachusetts Institute of Technology
RANSAC	Random Sample Consensus
PnP	Perspective-n-Point
ROS	Robot Operating System [5]
TF	Transformation

This page intentionally left blank.

REFERENCES

[1] A.S.P.D. Ashish Sharma, Mukesh Agarwal, "Motion capture process, techniques and applications," *International Journal on Recent and Innovation Trends in Computing and Communication* 1(4), 251–257 (2013).

[2] "tf - ros wiki," URL http://wiki.ros.org/tf.

[3] P. Milgram, H. Takemura, A. Utsumi, and F. Kishino, "Augmented reality: A class of displays on the reality-virtuality continuum," in *Photonics for Industrial Applications*, International Society for Optics and Photonics (1995), pp. 282–292.

[4] R. Azuma, "Tracking requirements for augmented reality," *Commun. ACM* 36(7), 50–51 (1993), URL http://doi.acm.org/10.1145/159544.159581.

[5] "Documentation - ros wiki," URL http://wiki.ros.org/.

[6] O. Oyekoya, R. Stone, W. Steptoe, L. Alkurdi, S. Klare, A. Peer, T. Weyrich, B. Cohen, F. Tecchia, and A. Steed, "Supporting interoperability and presence awareness in collaborative mixed reality environments," in *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology*, New York, NY, USA: ACM (2013), VRST '13, pp. 165–174, URL http://doi.acm.org/10.1145/2503713.2503732.

[7] S. Kasahara, R. Niiyama, V. Heun, and H. Ishii, "extouch: Spatially-aware embodied manipulation of actuated objects mediated by augmented reality," in *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction*, New York, NY, USA: ACM (2013), TEI '13, pp. 223–228, URL http://doi.acm.org/10.1145/2460625.2460661.

[8] V. Lepetit and P. Fua, "Monocular model-based 3d tracking of rigid objects," *Found. Trends. Comput. Graph. Vis.* 1(1), 1–89 (2005).

[9] P. Azad, D. Munch, T. Asfour, and R. Dillmann, "6-dof model-based tracking of arbitrarily shaped 3d objects," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on* (2011), pp. 5204–5209.

[10] P. Azad, T. Asfour, and R. Dillmann, "Accurate shape-based 6-dof pose estimation of single-colored objects," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, IEEE (2009), pp. 2690–2695.

[11] K.W. Chia, A. Cheok, and S. Prince, "Online 6 dof augmented reality registration from natural features," in *Mixed and Augmented Reality, 2002. ISMAR 2002. Proceedings. International Symposium on* (2002), pp. 305–313.

[12] "Opencv — opencv," URL http://opencv.org/.

[13] "Pcl - point cloud library (pcl)," URL http://pointclouds.org/.

[14] H. Fang, S. Ong, and A. Nee, "A novel augmented reality-based interface for robot path planning," *International Journal on Interactive Design and Manufacturing (IJIDeM)* 8(1), 33–42 (2014), URL http://dx.doi.org/10.1007/s12008-013-0191-2.

[15] "Artoolkit home page," URL http://www.hitl.washington.edu/artoolkit/.

[16] M. Simoes and C.G.L. Cao, "Leonardo: A first step towards an interactive decision aid for port-placement in robotic surgery," in *Proceedings of the 2013 IEEE International Conference on Systems, Man, and Cybernetics*, Washington, DC, USA: IEEE Computer Society (2013), SMC '13, pp. 491–496, URL http://dx.doi.org/10.1109/SMC.2013.90.

[17] G. Schweighofer and A. Pinz, "Robust pose estimation from a planar target," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28, 2024–2030 (2006).

[18] A. Cirulis and K.B. Brigmanis, "3d outdoor augmented reality for architecture and urban planning," *Procedia Computer Science* 25(0), 71 – 79 (2013), URL http://www.sciencedirect.com/science/article/pii/S1877050913012143, 2013 International Conference on Virtual and Augmented Reality in Education.

[19] F. Leutert, C. Herrmann, and K. Schilling, "A spatial augmented reality system for intuitive display of robotic data," in *Proceedings of the 8th ACM/IEEE International Conference on Human-robot Interaction*, Piscataway, NJ, USA: IEEE Press (2013), HRI '13, pp. 179–180, URL http://dl.acm.org/citation.cfm?id=2447556.2447626.

[20] M. Gianni, G. Gonnelli, A. Sinha, M. Menna, and F. Pirri, "An augmented reality approach for trajectory planning and control of tracked vehicles in rescue environments," in *Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on* (2013), pp. 1–6.

[21] "Kinect sensor," URL http://msdn.microsoft.com/en-us/library/hh438998.aspx.

[22] "rosbag - ros wiki," URL http://wiki.ros.org/rosbag.

[23] "Handle for tango project by lefabshop - thingiverse," URL http://www.thingiverse.com/thing:287481.

[24] "Trossen robotics," URL http://www.trossenrobotics.com/.

[25] "25.3. unittest unit testing framework python 2.7.8 documentation," URL https://docs.python.org/2/library/unittest.html.

[26] "unittest - ros wiki," URL http://wiki.ros.org/unittest.

[27] "11.4. shelve python object persistence python 2.7.8 documentation," URL https://docs.python.org/2/library/shelve.html.

[28] OLogic, "Rostango - ros on project tango," URL https://github.com/ologic/Tango.

[29] S. Suzuki et al., "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing* 30(1), 32–46 (1985).

[30] T. Kojima, K. Saito, T. Kakai, Y. Obata, and T. Saigusa, "Circularity ratio," *Okajimas Folia Anatomica Japonica* 48(2-3), 153–161 (1971).

[31] M.A. Fischler and R.C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM* 24(6), 381–395 (1981), URL http://doi.acm.org/10.1145/358669.358692.

[32] C. Cooper, "Math300 - chapter04: Cross ratio," Macquarie University, URL http://web.science.mq.edu.au/~chris/geometry/CHAP04%20Cross%20Ratio.pdf.

[33] X.S. Gao, X.R. Hou, J. Tang, and H.F. Cheng, "Complete solution classification for the perspective-three-point problem," *IEEE Trans. Pattern Anal. Mach. Intell.* 25(8), 930–943 (2003), URL http://dx.doi.org/10.1109/TPAMI.2003.1217599.

[34] N. Livet, "P3p (perspective-three-point): an overview — iplimage," URL http://iplimage.com/blog/p3p-perspective-point-overview/.

[35] P.J. Besl and N.D. McKay, "A method for registration of 3-d shapes," *IEEE Trans. Pattern Anal. Mach. Intell.* 14(2), 239–256 (1992), URL http://dx.doi.org/10.1109/34.121791.

[36] N. Ho, "Finding optimal rotation and translation between corresponding 3d points — nghia ho," URL http://nghiaho.com/?page_id=671.

[37] "Camera calibration and 3d reconstruction opencv 2.4.9.0 documentation," URL http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#solvepnp.

[38] A. Rebguns, C. Jorgensen, and C. Slutter, "Dynamixel controller package," URL http://wiki.ros.org/dynamixel_controllers.