SENSOR FUSION TO DETECT SCALE AND DIRECTION OF GRAVITY IN MONOCULAR SLAM SYSTEMS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Seth C. Tucker

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

December 2017

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Mohamed El-Sharkawy, Chair

Department of Engineering and Technology

Dr. Brian King

Department of Engineering and Technology

Dr. Paul Salama

Department of Engineering and Technology

Approved by:

Dr. Brian King

Head of the Graduate Program

To my wife, Niki, who put up with this whole thing and paid for it.

TABLE OF CONTENTS

Page

LIST OF FIGURES

ABSTRACT

Tucker, Seth C. M.S.E.C.E., Purdue University, December 2017. Sensor Fusion to Detect Scale and Direction of Gravity in Monocular Slam Systems. Major Professor: Mohamed A. El-Sharkawy.

Monocular simultaneous localization and mapping (SLAM) is an important technique that enables very inexpensive environment mapping and pose estimation in small systems such as smart phones and unmanned aerial vehicles. However, the information generated by monocular SLAM is in an arbitrary and unobservable scale, leading to drift and making it difficult to use with other sources of odometry for control or navigation. To correct this, the odometry needs to be aligned with metric scale odometry from another device, or else scale must be recovered from known features in the environment. Typically known environmental features are not available, and for systems such as cellphones or unmanned aerial vehicles (UAV), which may experience sustained, small scale, irregular motion, an IMU is often the only practical option. Because accelerometers measure acceleration and gravity, an inertial measurement unit (IMU) must filter out gravity and track orientation with complex algorithms in order to provide a linear acceleration measurement that can be used to recover SLAM scale. In this thesis, an alternative method will be proposed, which detects and removes gravity from the accelerometer measurement by using the unscaled direction of acceleration derived from the SLAM odometry.

1. INTRODUCTION

Monocular simultaneous localization and mapping (SLAM) algorithms are capable of inexpensively generating highly accurate maps of their environment and telemetry for agents within that environment. However, monocular SLAM can only report this information in an arbitrary scale. Monocular SLAM is far more useful if its arbitrary internal scale can be related to objective units of measure, such as meters. Given sufficiently accurate telemetry from corroborating sensors, this is easily achieved; a scaling constant can be determined by dividing telemetry measurements generated by the monocular SLAM algorithm by corroborating telemetry measurements of known metric scale. However, if extremely accurate redundant telemetry is available, than the telemetry obtained from monocular SLAM is unnecessary. The goal, therefore, is to find a scale factor with minimal additional sensor information in order to take advantage of the relatively inexpensive nature of monocular SLAM telemetry. Solving this problem is a topic of active research. This thesis will discuss some of these methods, and propose a novel new technique which has some advantages over existing methods.

1.1 SLAM

SLAM, or simultaneous localization and mapping, is a term used to describe the process by which an agent uses a sensor or sensors to map out its environment and its position in that environment. SLAM has its origins in robotics and it is one of the most important areas of research in the field. Robots, especially autonomous robots, need to know where they are, and what their environments looks like in order to be useful. Sophisticated autonomous robots often operate in environments that are not known ahead of time, so they need a way to determine this information on the fly

with their sensors [1]. Self-driving cars are one important example. A self-driving car controller needs detailed, accurate information about the road, road conditions, other vehicles, and pedestrians in order to safely pilot the vehicle. As researchers try to develop robots that interact with the world with greater autonomy, improved SLAM techniques will continue to be a key enabling technology.

SLAM has also found applications outside robotics. Virtual reality and augmented reality need detailed telemetry information in order to map real world motion to virtual world motion. A smart phone augmented reality app, for example, must used its sensors to align computer generated geometry to inferred real world geometry and then project it on the video feed from the camera. When the SLAM odometry or mapping is inaccurate, the virtual overlay jumps and slides around, breaking the illusion [2]. For virtual reality systems, odometry must be even better, as subconscious processes in the brain can be thrown off by errors that can not be consciously noticed, causing illness [3].

Three dimensional scanning is another application where SLAM is important. Visual SLAM techniques are used to generate models of places or objects which can be placed a virtual world, or 3D printed [4].

SLAM techniques can incorporate a wide variety of different sensors. LIDAR, sonar, wheel rotary encoders and inertial sensors are all commonly used for SLAM. Any device that uses sensors to develop a model of its environment is performing SLAM, and depending on the application, this may be very simple or very complex.

Visual SLAM is a term used to describe SLAM techniques that use cameras and extract 3D information from the images they take. There are two major types of visual SLAM. Stereo SLAM, which uses two cameras, and monocular SLAM, which uses only one. The followings sections will give a brief overview of the visual SLAM techniques, and provide some context for the motivation of this thesis.

1.1.1 Stereo SLAM

To understand monocular SLAM, it is helpful to first understand how stereo SLAM works. Stereo SLAM works similarly to our own natural stereoscopic depth perception. Stereo SLAM uses two cameras, separated by a known distance. Through direct image alignment on all pixels in the image, or by tracking features, the parallax between the two images can be used to infer the distance from the cameras to features or points in the environment and the camera.

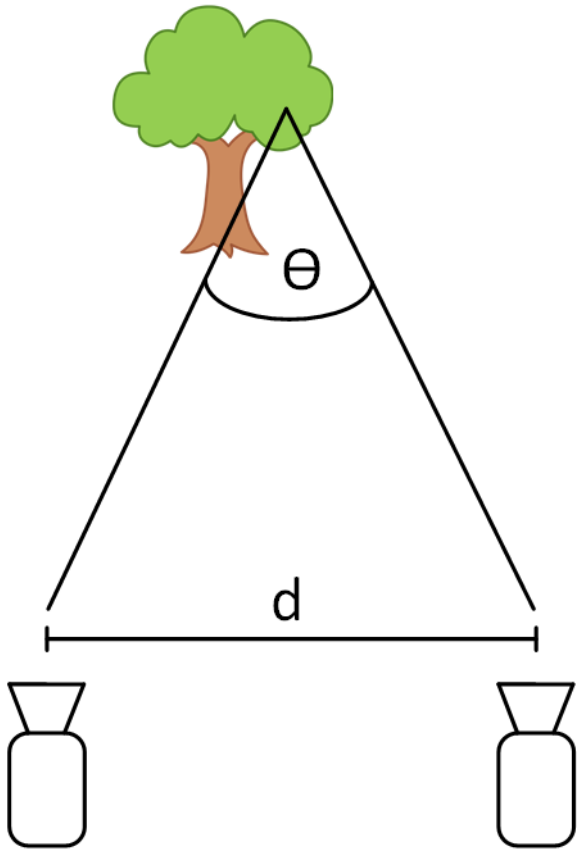


Fig. 1.1. The stereo parallax d, determines the size of the angle θ for a point at a given distance.

The two cameras make a triangle with a given point in the image. For distances that are within an order of magnitude of the distance between the cameras, the distance to a point can be determined with great accuracy. However, as the point

gets farther away, the parallax becomes small and the uncertainty grows extremely large [1]. This is intuitive, because it is exactly how we experience depth. We can accurately estimate the distance between objects that are within a few feet of us, but distant mountaintops may appear to be part of a single ridge, even though one may be twice as distant as the other.

1.1.2 Monocular SLAM

Monocular SLAM, as is indicated by the name, uses only a single camera. To achieve the image parallax necessary for inferring depth, it uses images taken at different points of time. As the camera moves, the system estimates change in the camera's position with a six degree of freedom rigid body transform. With the estimated change in orientation and position, it is possible to infer the depth in a manner similar to stereo SLAM. This has two major consequences. First, the camera position and the inferred depth are expressed in a arbitrary scale. Second, because a rigid body transform must be estimated every time, the error in that estimate accumulates and the map becomes inconsistent over time. Even the scale does not remain consistent. On looped paths, it is common to see the same features displaced and shown in significantly different sizes. In addition to scale and drift issues, monocular SLAM can only cope with camera movement that allows for sufficient parallax between frames. A stationary camera, or excessive rotational movement tends to cause a loss in tracking [5] [6].

Monocular SLAM has significant advantages and disadvantages in comparison to stereo SLAM. Monocular SLAM is effective in much larger environments than stereo SLAM, because camera movement can generate far greater parallax than the largest practical stereo configuration. It also does not require careful alignment and calibration of two cameras, which greatly reduces size and cost. However, because it uses time and motion to generate parallax, it is particularly unsuited to dynamic environments. It also means that the system can lose tracking if camera motion rotates

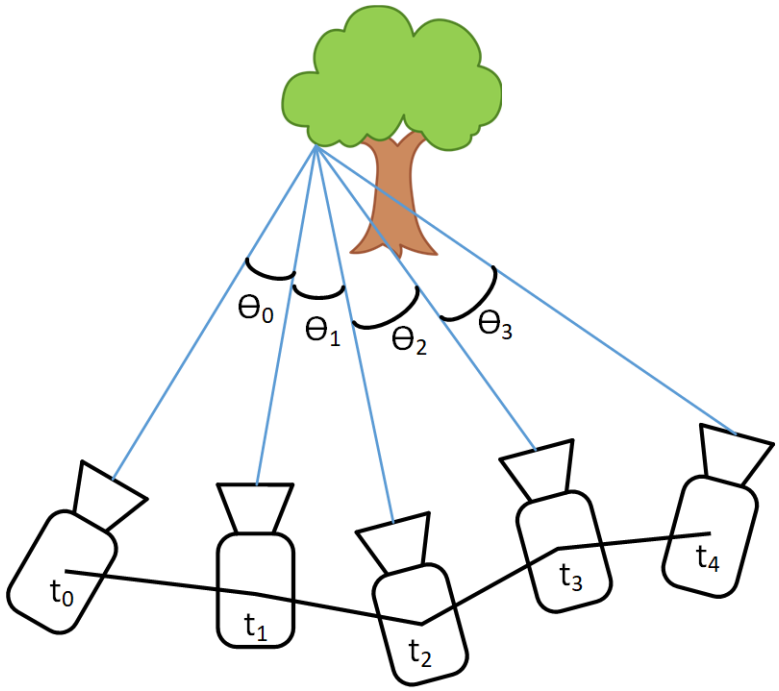


Fig. 1.2. Instead of using two cameras, monocular SLAM uses two images taken at different points in time on the same camera. Sufficient camera translation is required to generate parallax.

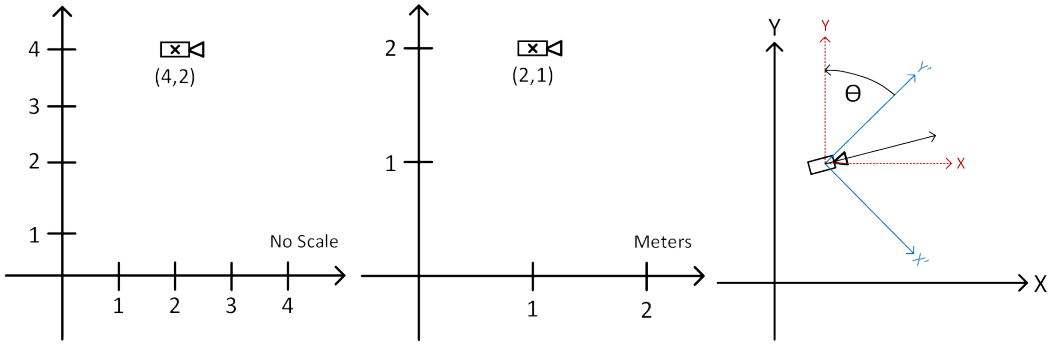
too much without generating any parallax from translational movement. Finally, the lack of known metric scale makes monocular SLAM systems susceptible to an extra degree of freedom of map drift, and prevent it from using its telemetry in conjunction with other sensors.

One important technique that monocular SLAM algorithms use to handle scale drift is loop closure [7]. As the map is created, often the same parts of the environment will be mapped several times. If the path of the camera takes it by features that have already been mapped, its path makes a loop, and if the scale or tracking estimate has drifted too much, it may result in another copy of same feature offset in the map. Loop closure analyzes nearby features already in the map and compares them to newly generated ones to determine if the camera is mapping the same feature twice. If it is, it will adjust the scale and path of the map all along the loop in order to make the map consistent.

This can greatly improve the consistency of the map, but it is not perfect. Not every camera path will make a loop, and excessive drift will make it impossible for loop closure to be detect [5].

1.2 Finding Scale and Orientation of Monocular SLAM

Monocular SLAM can be far more useful if its internal coordinate system can be referenced to the real world. In order to do this it is necessary to find two things. The first is a scale factor that can be used to convert distance values in the SLAM coordinate system to real world units. The second is a rotation that indicates how the SLAM world aligns with absolute geographical coordinates.



(a) SLAM Scale (b) Meter Scale (c) World oriented axes X and Y, and SLAM oriented axes, X' and Y'

Fig. 1.3. Analogy for map scale and orientation. In this case the scale factor is 0.5, and the camera coordinates are rotated by 45 degrees compared to the world coordinates.

If the scale factor and the relative orientation can be found, many of the drawbacks of monocular SLAM can be mitigated. For example, the algorithm can be adapted to recognize and correct scale drift.

In addition, it makes it possible to use the SLAM telemetry in conjunction with other sensors to improve position and orientation estimates. It also makes it possible to use SLAM telemetry for navigational navigation or control purposes.

There are two basic methods that can be used to determine scale. One method is similar to how a human would estimate the size of distant object with only a single eye; recognized features in the environment with known size are used to infer the scale of the map. The other method consists of comparing SLAM telemetry to coincident telemetry from independent sensors that and calculating a ratio between the two.

The process for finding the absolute orientation of monocular SLAM is no different than it is for any other system. A sensor must be used to measure the angles between the coordinate system and geographic coordinates, or else the system must be initialize to a known orientation relative to those coordinates. Depending on the application, the absolute orientation may not need to be fully determined, but often it is at least necessary to find the direction of gravity.

1.2.1 Scale from Image Recognition

It is common to use known features or landmarks in the environment to establish scale. A printed grid of known size can be used to infer scale as well as orientation and relative position with a single camera. This implementation is common in marker based augmented reality systems such as ARToolkit [8]. Marker based augmented reality isn't really SLAM, but markers can be used in conjunction with SLAM to provide scale. Unfortunately, most real world applications do not involve environments that contain known markers to detect scale. As image recognition improves, it will become possible to infer scale from a wider of objects found in the environment, rather than relying on artificial markers, and some efforts have been made to do this [9]. However, these techniques are still limited to environments which have features of approximately known size.

1.2.2 Sensor Measurement Considerations

In order to understand the challenges associated with using other sensors to establish SLAM scale, it is helpful to discuss the different frames of reference that sensors use, and how this effects their short and long term accuracy.

We are interested in two basic kinds of measurements; spatial, and orientational. These sensors measure distance and orientation, or their derivatives. Spatial sensors measure position, velocity or acceleration. Orientation sensors measure orientation, or angular rate. Typically, it is not possible to make a direct comparison between two measurements of the same quantities. Sometimes quantities must be compared to with their associated rate, or to a value indirectly inferred from several other sensors.

To get a direct comparison between a measured quantity and a measurement of that quantity's rate, it is necessary to either take the derivative of that quantity, or integrate the rate. Care must be taken in choosing which approach to take, as data integration increases accumulated error, but discrete differentiation can greatly increase signal noise [10]. Generally, integration is easier to implement, but is not always viable, as it requires an initial estimate. For example, it is possible to sum velocity measurements to find position, but only if you know the position associated with the first velocity in the sum. Essentially, integration increases accumulated drift, while differentiation will increase noise. Depending on the sensor, and the application, one may be clearly preferable.

Sensors do not all measure in the same coordinate frame. A GPS, for example, will give velocity or position relative to the earth in geographic coordinates. Monocular SLAM does not give position relative to geographic coordinates, but does give it with respect to a fixed location and orientation in the world. An accelerometer, on the other hand, measures acceleration in a frame that rotates with the accelerometer. In order to compare two odometry values which are in different coordinate frames, it is necessary to transform one of the values into the other's coordinate frame.

Precision and stability are also very important. GPS has very low precision of about 4 meters. Compared to an inertial measurement unit, this is very low, but because it makes a direct measurement of position, its precision is constant for every reading. An inertial measurement unit (IMU) may provide very detailed telemetry, but because it has to estimate location by summing rate measurements, the uncertainty in its absolute position estimate drifts at a quadratic rate [11]. It may be able to measure short distances with centimeter precision, but over a longer distances, the position estimate will grow until it becomes even worse than the one given by GPS. The type of measurement uncertainty is important as well. If measurements are only corrupted by noise with a known statistical distribution it is possible to at least know how accurate the information is. However, instruments like the compass can have their readings distorted by unpredictable local distortions in the magnetic field. Sources of error like this cannot easily be accounted for, and present a larger problem than statistically known error.

Sample rate is another concern. Higher sample rates allow for a more detailed estimates of camera motion. A high precision is not as useful if it is reported slowly, because it is only possible to guess what happens between samples. Generally sample rates will not be synced up, so it will be necessary to interpolate between two samples in order to compare two values at the same point in time. A high sample rate will make that interpolation more accurate.

1.2.3 Scale from Corroborating Sensors

Monocular SLAM reports its telemetry in terms of an orientation and a position in a reference frame whose origin is the initial pose. In other words, it will take the origin point to be the location of the camera at the first frame of the video feed, and the axes will be aligned with the camera's orientation at that point. To find the scale factor for the SLAM, a sensor must provide a co-witness for the camera's motion and the two measurements must be compared in the same reference frame. An ideal set

of sensors for scaling the SLAM map would measure the angles between the direction of the camera and geographic north and the gravity vector. It would also report the absolute position of the camera in coordinate that are either aligned with the initial position of the camera or in geographic coordinates. The combination would make it easy to know the camera's position and orientation in six degrees of freedom. Finding scale would be a simple matter of dividing change in position in meters by the change in position in the SLAM coordinate system. Any pose or distance in the SLAM map could be expressed in metric scale and in geographic coordinates by using a rotation matrix to transform orientation, and multiplying by the scale factor.

Unfortunately, sensor packages that can provide an accurate gravity vector, geographic heading and absolute position are extremely expensive, and obviate much of the advantages of an inexpensive monocular SLAM system. Any given set of sensors will likely only be financially or functionally viable for limited subset of usage scenarios. Greater numbers of sensors can give the system greater functional flexibility at the expense of cost and complexity.

System motion plays a big part in determining what kinds of sensors will be most useful. Imagine a small 10cm by 10cm wheeled robot navigating a flat course in a small room. A vehicle that travels only across flat ground can assume the direction of gravity a priori. This way it only needs to determine heading, and two degrees of positional motion. Magnitude of direction is important as well. For example, standard uncorrected GPS, in addition to being nonfunctional indoors, has a precision of about 4m, which is larger than the robot's entire environment. This is far too great an uncertainty to make any useful estimation of scale. At best, you could establish an upper limit on the scale factor. However, if the robot had rotary encoders on its wheels, it could measure how far they've turned and estimate it's position to a high degree of precision. The rotary encoder's position estimate would eventually drift due to wheel slip and accumulated error, but as long as the scale is continually updated, the loop closure in the SLAM map would provide some degree of correction.

A priori information about the system can make scale estimation far easier, but it is not always available. In the most general case, a system can have three degrees of positional freedom and three degrees of orientational freedom. For the purposes of establishing scale, it is sufficient to have all three dimensions of position measured, but often it is necessary to have an orientation sensor to align the coordinate system of the spatial sensors with the SLAM coordinates.

1.2.4 Overview of Major Sensor Types

In order to successfully implement a telemetry sensor, it is important to take into account not just what it measures, but also how it works. Sensors are not magic, and their physical realization must be understood in order to know if they suit their intended application. A single type of sensor may even have a number of different implementations which behave in significantly different ways. It is not possible to give a detailed account of all the possibilities here, but a short description of some of the options will give greater insight into the overall problem.

GPS GPS has low precision of about 4 meters, and a low sample rate of around a second, but it directly measures position referenced to geographic coordinates, so its positional error does not grow over time. GPS is far less precise for altitude measurements, so it is best to use it in conjunction with a barometer.

Barometer A barometer infers altitude from an atmospheric pressure measurement, but does so reasonably quickly to a precision as good as 0.3 meters. However, it is highly sensitive to weather conditions which can cause significant absolute inaccuracy if not constantly calibrated to a known altitude reference. Without adjustment it can provide reasonably accurate relative measurements over short periods of time.

Compass A compass measures the earth's magnetic field and can provide a bearing relative to magnetic north. With a rough estimate of global location, this can

be adjusted to find orientation relative to true North. A 3 axis magnetometer compass can also measure tilt in order to give the direction of gravity. While inexpensive microelectromechanical system (MEMS) compasses can achieve a measurement precision of less than a degree under ideal circumstances, local fluctuations in the magnetic field due to metals and electronics can significantly and unpredictably distort the measurement.

Gyroscope A gyroscope measures angular rate, or orientation depending on the implementation. Rate gyroscopes can be integrated to determine orientation relative an initial position. Gyroscopes vary widely in precision and accuracy. Ring laser gyroscopes used in military inertial measurement systems are accurate enough to measure the rotation of the earth and can be used to track orientation for very long periods without significant drift. They can even be used for submarine navigation in place of a compass. At the other end of the spectrum, MEMS gyroscopes are small and inexpensive, but are far less accurate and orientation estimates will drift significantly over periods longer than a few seconds. The high sampling rate of gyroscopes make them useful for interpolating between lower speed orientation measurements.

Accelerometer An accelerometer measures inertial force. Three mutually orthogonal accelerometers are used to measure the direction and magnitude of inertial force in three dimensions. In a non-inertial reference frame like the earth's surface, it measures the vector of acceleration due to motion plus the vector of force due to gravitational acceleration. Without knowing the direction of gravity in the accelerometer's reference frame, and without other information about the motion of the accelerometer, it is not possible to know the magnitude or direction of true acceleration or the direction of gravity. The gravity vector and the true acceleration vector add, so the true acceleration is the measured vector minus the gravity vector. The magnitude of gravity on the earth's surface is known a priori, which means that the true acceleration vector lies on somewhere on 1g sphere around the measured acceleration vector.

Inertial Measurement Unit An Inertial Measurement Unit (IMU) is a system that combines the output several of inertial sensors to provide a more complete set of telemetry data than single sensor could alone. A typical low cost MEMS sensor such as those found on a smart phone commonly use a three axis magnetometer, a three axis accelerometer and a three axis gyroscope to provide orientation, angular rate, and true acceleration. The compass tilt sensor is used to estimate the direction of gravity. This allows the gravity vector to be subtracted from the accelerometer measurement. The gyroscope can be integrated over short period to compensate for local magnetic field anomalies. While military IMUs are accurate enough to be used for dead reckoning navigation over long distances, the more inexpensive IMUs typically used in low cost robots and smartphones drift too quickly to be use alone for navigation. Positional error grows quadratically, and the system is susceptible to magnetic field disturbances. IMU navigation can be made far more useful in conjunction with visual SLAM, which drifts at a much lower rate and can take advantage of loop closure.

1.3 Minimal Accelerometer Based Scale

It should be clear at this point that, given the correct sensors and a priori assumptions, it is fairly easy to find monocular SLAM scale. But generally, we want to get the best possible result from the fewest possible sensors and assumptions. For large scale outdoor applications, GPS can do a good job, and implementation is easy. But what about medium and small scale, and what if GPS is not available? What if we can't make a priori assumptions about the initial position, motion or orientation of our system? There are not a lot of good options for cheaply and robustly detecting position with high resolution. In fact, monocular SLAM is of great interest precisely because it can do just that- if we can also find scale equally cheaply.

This thesis will show that monocular SLAM scale and the direction of the gravity vector can be determined with only a MEMS three axis accelerometer, a MEMS three axis gyroscope, and the a priori assumption that the camera is not tilted more than 90 degrees away from the gravity normal vector. It will also demonstrate an attempt to calculate the covariance matrix of the result from the covariance matrices of the inputs, and some of the limitations of this approach. This solution is significant because it allows the gravity vector and scale to be determined without the aid of a compass which is susceptible to unpredictable sources of error.

1.4 Overview

First, we will explore the theoretical background and justification for the proposed scaling method, covering the conceptual background of the ideas presented, then solving the solution for the case of ideal odometery. Then we will show alternative solutions for inconstant odometry, and then deal with the physical separation of sensors on the system. Finally, we will demonstrate a way to calculate linearized variance estimate for the scale.

Next, we will deal with the implementation of the system covering the hardware, software interfaces, the monocular SLAM algorithm and the implementation code.

The results of system tests will be shown and discussed, followed by concluding remarks, and a short discussion of areas in which the ideas proposed here should be developed in future work.

2. METHOD

This chapter will describe a method for determining the scale factor and gravity vector for a monocular simultaneous localization and mapping (SLAM) system using inertial sensor data together with SLAM odometry. Provided a sufficient magnitude of acceleration, it can initialize from any state provided its orientation is tilted less than 90 degrees from the gravity normal vector. And, with minor modification, even this limitation can be overcome. This allows the system to inexpensively establish scale and gravity direction in small scale environments while undergoing erratic motions that are challenging or impossible for other methods to deal with. While it may be easier to scale monocular SLAM with other sensors, no one sensor is capable of accurately providing scale in every situation. The method proposed here helps solve one of the most difficult scenarios for SLAM scaling.

2.1 Notation

In the most general case, monocular SLAM can be expected to work in systems which experience arbitrary 3D motion. To be as general as possible, no system dynamic model can be used to predict system position or motion. System models vary with implementation, and some systems don't have them at all. A handhold cellphone is a particularly relevant example. For the purposes of this thesis, we will consider a 15-dimensional state, $\vec{\zeta}$, to describe the position, motion and orientation of the camera.

$$\vec{\zeta} = (x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}, \ddot{x}, \ddot{y}, \ddot{z}) \tag{2.1}$$

This state consists of the position, velocity and acceleration of the camera along x, y and z axes, as well the angles ϕ, θ and ψ which represent the roll pitch yaw orientation angles, and angular rates. Note that in practice, quaternions or some other

method will be used to represent rotations in lieu of roll, pitch and yaw angles. These alternative methods are much less intuitive, but they have the important advantage of avoiding gimbal lock.

Monocular SLAM odometery reports the camera's position with each new camera frame, in a fixed coordinate system whose origin is usually referenced to the position and orientation of the first camera frame. An accelerometer, on the other hand, is mounted rigidly to the camera, and it will measure a three dimensional acceleration vector in the body coordinate frame, which is rigidly attached to the camera. To work with these measurements, we will need to rotate one of them into a coordinate frame aligned with the other. If we use a known rigid transformation to express an accelerometer reading in a coordinate frame aligned with the SLAM coordinates, we will say that it is in the SLAM coordinate frame, even though technically it is only in a set of coordinate frames that share a common orthonormal basis.

We will use superscripts to denote reference frames that share a common alignment. To refer to a state variable that is expressed in a coordinate system aligned with the SLAM coordinate system, we will use the superscript 's' to represent 'SLAM'. For example, the full state in SLAM aligned coordinates would be $\vec{\zeta}^s$. On the other hand, an accelerometer measurement is taken in the body frame, so this measurement can be represented by the vector $\vec{a^b}$, though it is important to remember the transformation between a body frame and a fixed frame changes with each new measurement, so not all "body" frames are aligned through time. At first, we will assume that the physical separation between the camera and the accelerometer is small enough to be ignored, so both the camera and the accelerometer will share a single body coordinate frame.

Finally, a subscript will generally refer to the instrument that the particular value was derived from. For example, $\vec{a_a^b}$ refers to the acceleration measured by the accelerometer in the body frame, whereas $\vec{a_s^s}$ refers to the acceleration according the SLAM odometery in the SLAM coordinate frame.

2.2 Inertial Solution

This thesis will propose a method for using an accelerometer to establish monocular SLAM scale in a non-inertial reference frame. In order to appreciate that, and in order to provide context, we will first examine how the problem would be approached in an inertial reference frame where an accelerometer would not be exposed to gravity. Disregarding gravity greatly simplifies the problem, and helps to demonstrate some of the signal and measurement problems that will need to be dealt with later. An inertial environment also allows for some attractive and convenient approaches that are not practical in a non-inertial environment. The reasons that they do not work in the non-inertial case are not always immediately obvious without a clear understanding of the differences between the inertial and non-inertial solutions.

Recall that the objective is to compare a measurement taken in the SLAM coordinate system to an equivalent measurement taken in metric units. We will accomplish this with SLAM odometery, and a MEMS accelerometer.

The SLAM odometry reports $\vec{p}^{s}$, a vector describing the camera position in SLAM coordinates. The accelerometer provides $\vec{a}^{b}$, a vector indicating camera acceleration in body coordinates. In order to compare these two values, we have to do two things. First, we must get both readings to have the same class of unit, meaning they need to both be either a position, rate or acceleration. Second, we must express both values in an aligned coordinate system.

To get both measurements to have the same unit type, we have three options. $\vec{a}^{b}$ can be twice integrated to get position; $\vec{p}^{s}$ can be twice differentiated to get acceleration; or $\vec{a}^{b}$ can be once integrated, and $\vec{p}^{s}$ can be once differentiated to work with velocity. Differentiation is challenging, and can amplify noise, especially if done twice. Integration, on the other hand, is implemented as a summing operation which causes error accumulation.

Typically SLAM will generate $\vec{p}^{s}_{s}$ at a much lower data rate than the accelerometer generates $\vec{a}^{b}_{a}$. Camera frame rates will typically be between about 40hz and 80hz,

while accelerometers can easily achieve a data rate of 500hz. However, low cost Microelectromechanical systems (MEMS) accelerometers can't reliably track position over any significant distance due to excessive drift. As we will see later, $\vec{p}^s$ is not a very noisy signal, but twice differentiating it will produce significant noise and require filtering over several samples, further limiting the the bandwidth of our slowest signal. By meeting in the middle, we can keep noise to an acceptable level, while keeping our integration interval small enough to avoid any significant drift.

To get $\vec{v}_s^s$, the velocity in the SLAM coordinates according to SLAM odometery, we can do a simple first order divided difference between $\vec{p}_0^s$ and $\vec{p}_1^s$, the SLAM odometery of two successive camera frames. Here, f_s is the frame rate of the SLAM camera.

$$\vec{v}_s^s = (\vec{p}_1^s - \vec{p}_0^s) f_s \tag{2.2}$$

The accelerometer signal, $\vec{a}_0^b ... \vec{a}_n^b$, must be integrated over the period between $\vec{p}_0^s$ and $\vec{p}_1^s$. However, we don't want to integrate the accelerometer signal in the body frame, because its orientation potentially changes with every data point. One solution to this is to put each $\vec{a}_i^b, i = 0, 1..n$ in the SLAM frame. In order to do this, we need to know the rotation between the body coordinate system and the SLAM coordinate system. We have the RPY angles between the SLAM coordinates and the body coordinates. These are $\vec{o^s}$, the orientation portion of the SLAM odometry. $\vec{o^s}$ consists of the elements $(\phi, \theta, \psi) \subset \vec{\zeta}^s$. We can map this to the rotation matrix R^{sb} with the following equation,

$$R^{sb} = R_z(\phi^s)\, R_y(\theta^s)\, R_x(\psi^s) \tag{2.3}$$

where R_x, R_y, R_z, are the three basic rotations. The rotation between the body coordinates and the SLAM coordinates is then,

$$R^{bs} = (R^{sb})^{-1} \tag{2.4}$$

For accelerometer data rates sufficiently higher than the SLAM camera frame rate, we can assume $\vec{o}_0^s$ approximately coincides with the next closest accelerometer measurement $\vec{a}_0^b$. For $\vec{a}_1^b ... \vec{a}_n^b$, we integrate the angular rate from a MEMS gyroscope that

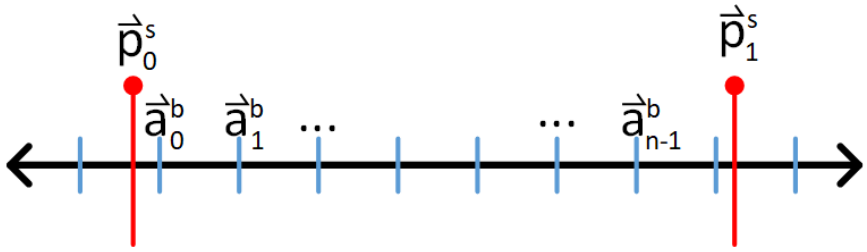


Fig. 2.1. The data rate of the accelerometer is faster than, and unsynchronized with the frame rate of the Camera being used for SLAM.

is synchronized with the accelerometer to find the roll pitch yaw angles between $\vec{a}_0^s$ and $\vec{a}_i^s$, for $i = 0, 1...n$. We then map these angles to rotation matrices $R_i^b, i = 1, 2...n$. These matrices represent the angle that the body has rotated between accelerometer measurements $\vec{a}_0^b$ and $\vec{a}_i^b$. For each $\vec{a}_i^b$, we find,

$$\vec{a}_i^s = R^{bs} R_i^b \vec{a}_i^b \tag{2.5}$$

where $\vec{a}_i^s$ is the accelerometer measurement in the SLAM coordinates. By integrating the signal $\vec{a}_0^s ... \vec{a}_n^s$ we obtain a vector $\vec{p_a^s}$, representing the distance between p_0^s and p_1^s according to the accelerometer.

To find the scale factor, we obtain $p_s^s = p_1^s$ - p_0^s, the vector between p_0^s and p_1^s according to the SLAM odometer. Finally, the scale factor is the ratio of the vector magnitudes: $S_f = ||p_a^s||/||p_s^s||$.

Notice that even though the plan was to work with velocities, it appears that we are working instead with position vectors. The single integration of $\vec{a}_a^s$ needs to be divided by the time elapsed over the interval to make the units correct. However, p_1^s - p_0^s only needs to be divided by the same time interval to be a valid first order discrete differentiation. The two time intervals are the same, so they can be ignored because they will cancel out. We avoid the drift issues of an actual position comparison, because we are only looking at the position vectors that represents change in position over a time interval, which is just another way of saying 'velocity'.

Sadly, this doesn't work unless you happen to be inside the International Space Station. Accelerometers do not measure linear acceleration, they measure linear acceleration plus the force of gravity [12]. To obtain acceleration from an accelerometer, you must know the direction of the gravity normal vector, $\vec{g}_n$ and subtract it from the accelerometer measurement $\vec{a}^b$.

$$\vec{a}^b_{true} = \vec{a}^b_a - \vec{g}_n \tag{2.6}$$

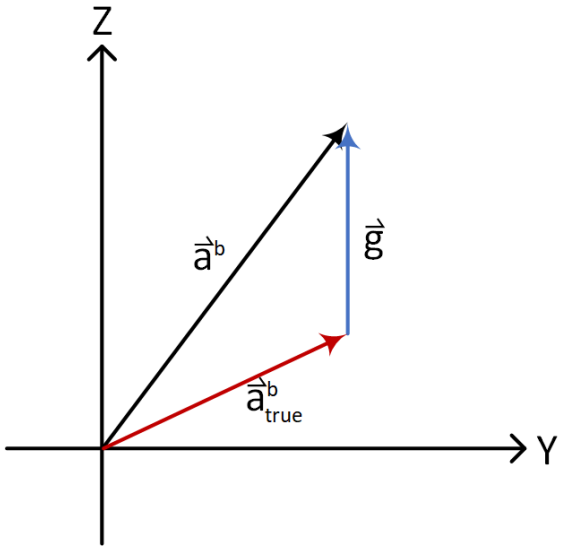


Fig. 2.2. The data rate of the accelerometer is faster than, and unsynchronized with the frame rate of the Camera being used for SLAM.

2.3 Non-Inertial Solution

Determining the direction of gravity can be fairly difficult to do robustly. There are a number of ways to do it, each with their own limitations. Typically a Kalman filter and other algorithms are used to combine several relatively poor estimates of the gravity vector [13]. If the system is known to not be undergoing linear acceleration, the accelerometer measures only the gravity vector. The gyroscope can track changes in orientation between these times. A high pass filter can be used remove the gravity

offset from the accelerometer over a period of time, and a 3 axis magnetometer with can provide a rough orientation estimate. None of these systems are particularly robust. Sustained acceleration may prevent a filter from obtaining an accurate gravity vector. Depending on the type of acceleration the device undergoes, the signal may have to be very aggressively filtered to get a good gravity estimate. The magnetic field needs local calibration information, and is highly susceptible to disturbance.

We will demonstrate a method for determining the linear acceleration component of an accelerometer measurement by comparing its direction to that of the linear acceleration derived from monocular SLAM odometry. The result easily leads to an estimate of the gravity vector and scale in a manner similar to the inertial solution outlined in the previous section. This solution is most robust under significant acceleration, and is well suited to initialization. It is a natural complement to traditional intertial measurement unit (IMU) gravity estimators as it has an inverse symmetry with those method's strengths and weaknesses. Finally, we will propose a linearized covariance estimate which can provide a rough relative estimate of accuracy sufficient for inverse-variance weighting of the scale.

At this point, it should be clear that, while ambiguous scale compromises any linear quantities derived from monocular SLAM odometry, no such problem exists for angular measurements. Scale is not defined for angular measurements and we have already established that we can align the reference frames of our inertial sensors to that of the SLAM map. Therefore, we can use the direction of linear vectors in the SLAM coordinate frame, even if the magnitudes are unscaled. Consider the unit vector of the twice differentiated SLAM odometery position vector. Recall that the superscript refers to the coordinate frame of the vector, and a letter subscript refers to the source of the vector, ie s for SLAM, or a for accelerometer.

$$\vec{a}_s^s(t) = \frac{\partial^2}{\partial t^2}\vec{p}^s(t) \tag{2.7}$$

$$\vec{u}^s = \frac{\vec{a}_s^s}{\|\vec{a}_s^s\|} \tag{2.8}$$

Note that in practice, $\vec{p}^s(t)$ is a discrete function rather than continuous, so its second derivative must be determined with a discrete approximation. We will discuss this in greater detail in a later section.

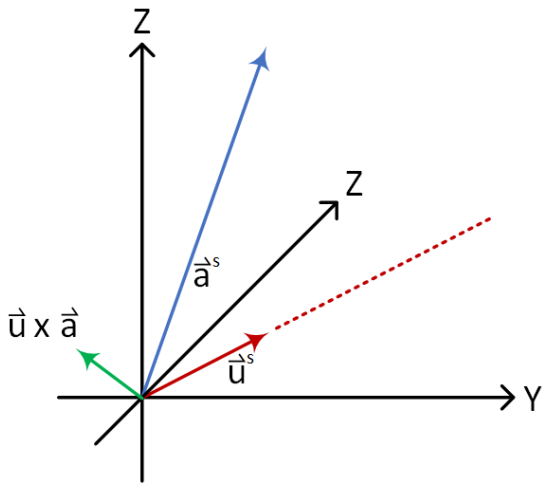


Fig. 2.3. $\vec{g}_n$ lies somewhere in the $\vec{u}^s \times \vec{a}_a^s$ plane.

Consider the $\vec{u}^s \times \vec{a}_a^s$ plane in Figure 2.3. Assuming accurate values for $\vec{u}^s$ and $\vec{a}_a^s$, the vector of scaled linear acceleration must lie somewhere along the direction of $\vec{u}^s$. If $\vec{u}^s$ and $\vec{a}_a^s$ are both accurate and consistent, we can assume the following will hold true, where $\vec{w}^s$ is the vector of scaled linear acceleration.

$$\vec{w}^s = \vec{a}_a^s - \vec{g}_n \tag{2.9}$$

This places that $\vec{g}_n$ somewhere in the $\vec{u}^s \times \vec{a}_a^s$ plane. If we can find $\vec{g}_n$, we can find $\vec{w}^s$, and from there get our scale. Happily, we know $||\vec{g}_n|| \approx 9.81 m/s^2$ at the earth's surface. Intuitively, it should be clear that if we trace a circle of radius $9.81 m/s^2$ from the end of the $\vec{a}_a^s$ vector, it will intersect a line extending from $\vec{u}^s$ at one or two points. These points are candidates for the linear acceleration vector $\vec{w}^s$. Because the scale value must be positive, $\vec{u}^s$, any accelerometer measurement with a magnitude of less $1g$, will give a unique solution for $\vec{w}^s$. However, if $||\vec{a}_a^s|| \geq 1g$, there are potentially two valid candidates for $\vec{w}^s$ and $\vec{g}$.

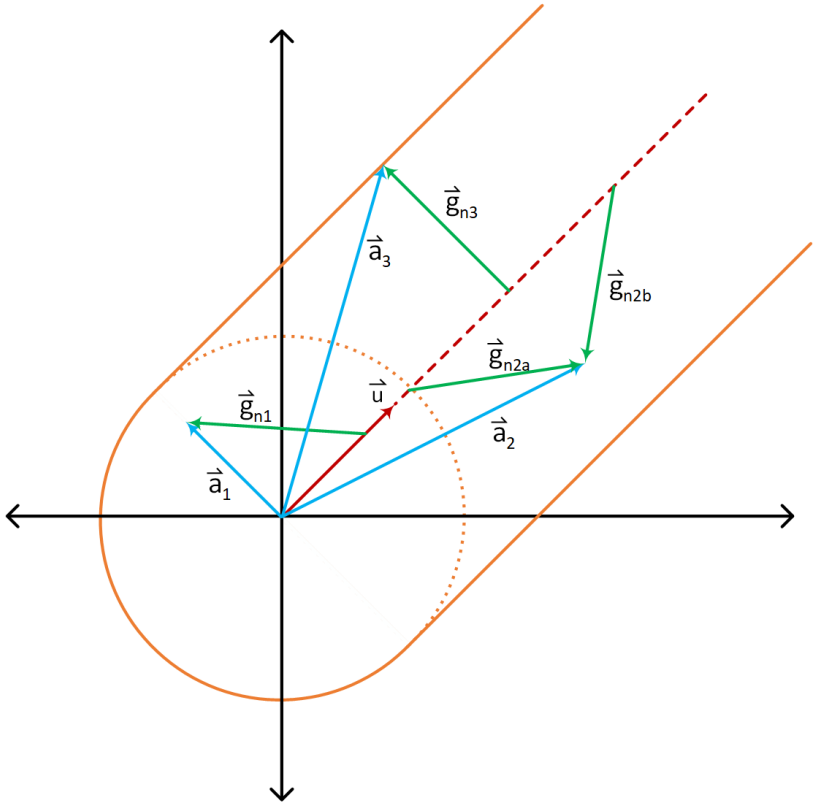


Fig. 2.4. For a given SLAM acceleration unit vector, $\vec{u}$, accelerometer measurement vectors can potentialy end anywhere within $9.81 m/s^2$ of the positive half of the $\vec{u}$ line. Depending on where $\vec{a}_a$ lies in that range, there may be either one or two possible vectors for gravity. If $\vec{a}_a$ ends inside the dash-lined sphere, like $\vec{a}_1$ or on the boundary lines, like $\vec{a}_3$, there is only one solution.

It's easy to see graphically that this is the case, but to understand how to choose the correct gravity vector, we need to conceptually grasp what the two different vectors represent. To make it easier, we will use the example of a one dimensional system where linear acceleration is constrained to lie along gravity vector. Imagine an accelerometer attached to a rocket, oriented in the direction of the rocket's nose. On the launch pad, it will read 1g. In order for it to accelerate upwards at 1g, the rocket motors must exert enough thrust to cause the accelerometer to measure 2g, 1g to counteract gravity and 1g of linear acceleration upwards. However, if the same rocket is instead launched directly toward the earth with the same thrust, the accelerometer

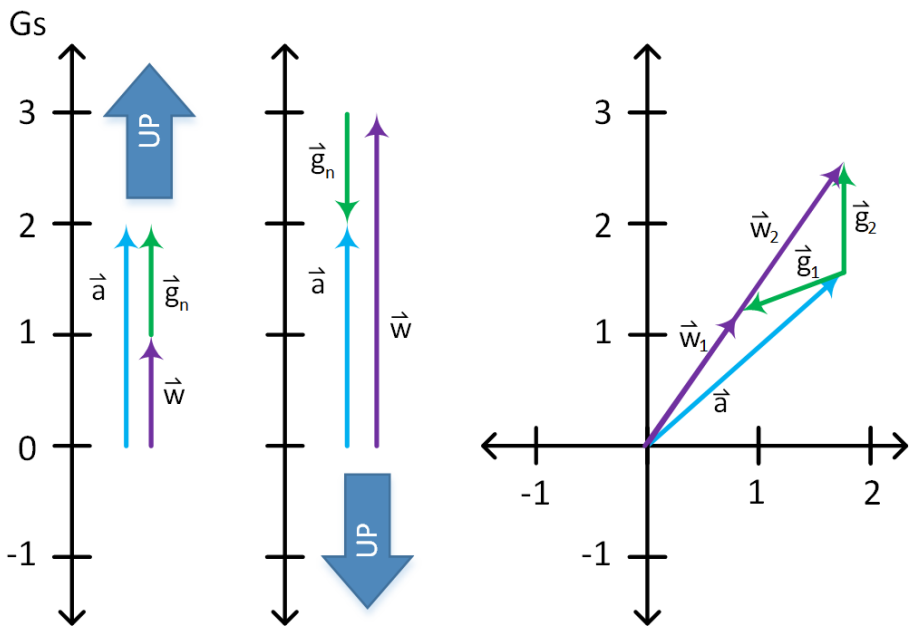


Fig. 2.5. The left and center diagrams represent the accelerometer reading in two different orientations.

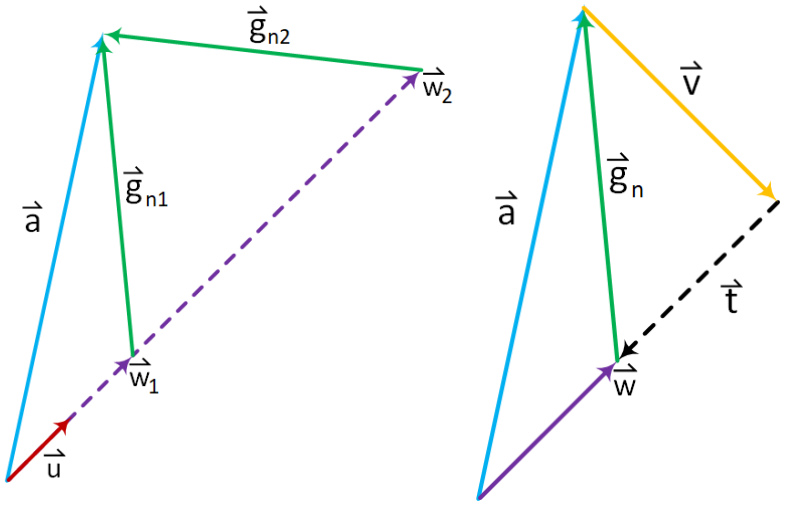
will still measure 2g, but this time the 1g of gravitational acceleration will add to this value, resulting in a linear acceleration of 3g.

We cannot determine which of these possibilities is correct with only the accelerometer and SLAM odomtery, but it's easy to see that the orientation does not have to be known to any significant degree of accuracy in discriminate between the two possibilities. In this case, we can decide as long as we have some sensor, or a priori guarantee that the system is either inverted or upright. While it may be very difficult to track the gravity vector with great accuracy using traditional methods, it is trivial to make a general distinction between a possible upright or inverted orientation.

This one dimensional perspective extended easily to two and three dimensions, as we can see from the right hand scenario in Figure 2.5. Because $\vec{a}_a^s$ and $\vec{u}^s$ are no longer aligned, the gravity vector candidates are no longer 180 degrees apart. As the angle between the gravity vectors decreases, it become progressively more difficult to distinguish between the two candidates. Fortunately, the error incurred by picking the

wrong vector for $\vec{g}_n$ progressively decreases as the difficulty rises, until at a separation of zero degrees, the two choices are equivalent. This corresponds to $\vec{a}_3$ in Figure 2.3.

2.3.1 Standard Computation of Linear Acceleration



(a) Both Solutions (b) Detail of Single Solution

Fig. 2.6. This illustrates the vectors which are used to solve for $\vec{w}$, the scaled acceleration. (b) illustrates the case where $\vec{t}$ is subtracted for the vector projection.

We can now describe a general method to calculate both candidates for metric linear acceleration, $\vec{w}$, given $\vec{u}$, $\vec{a}_a$, and the fact that $||\vec{g}_n|| \approx 9.81 m/s^2$. We will assume that all measurements are expressed commonly aligned coordinate frames. We have already defined $\vec{u}$ in equations 2.7 and 2.8. Next we define $\vec{v}$, the vector rejection of $\vec{a}_a$ onto $\vec{u}$,

$$\vec{v} := \vec{a}_a - (\vec{a}_a \cdot \vec{u})\vec{u} \tag{2.10}$$

Using the Pythagorean Theorem we find $\vec{t}$,

$$\vec{t} := \left(\sqrt{||\vec{g}|| - ||\vec{v}||}\right)\vec{u} \tag{2.11}$$

Finally we find the two candidates for $\vec{w}$ by subtracting and adding t from the vector projection of $\vec{a}_a$ onto $\vec{u}$,

$$\vec{w} = (\vec{a}_a \cdot \vec{u})\vec{u} \pm \vec{t} \tag{2.12}$$

2.3.2 Dealing with Inconsistent Measurements

Thus far, we have been operating under the assumption that acceleration measurements according to the accelerometer $\vec{a}_a$ and according to SLAM odometry, $\vec{a}$ are error free and consistent. In practice both measurements are susceptible to noise, and can sometimes become inconsistent. In Figure 2.4, the outside boundaries represent the region around a given $\vec{u}$ where there exist allowable terminations of the $\vec{a}_a$ vector. If $\vec{a}_a$ falls outside those boundaries, there is no possible $\vec{g}_n$ that can resolve the two measurements, and no solution exits. If we try to compute it anyway, we find that $||\vec{v}|| > ||\vec{g_n}||$, and $\vec{t}$ becomes an imaginary vector.

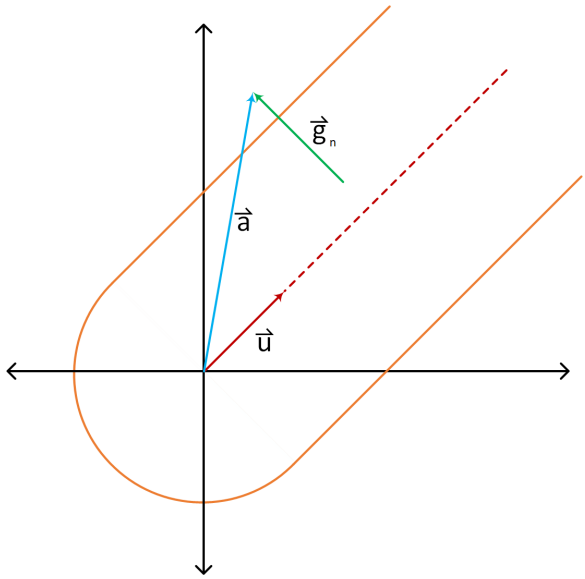


Fig. 2.7. When $\vec{a_a}$ lies outside the boundary, $\vec{g}$ cannot reach any scalar multiple of $\vec{u}$, and no solution exists.

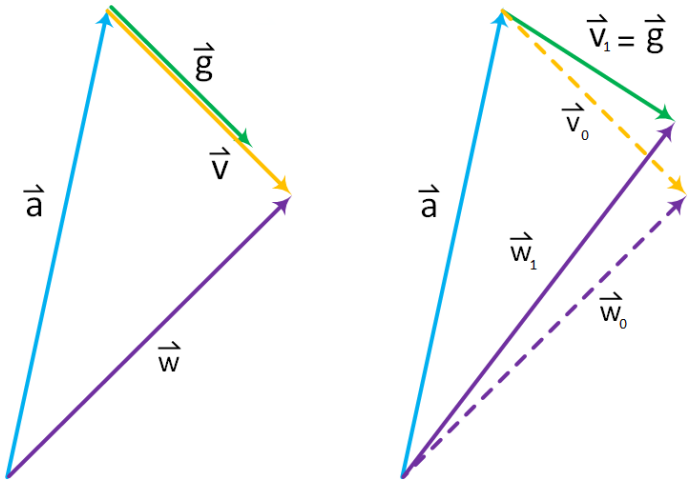
In practice, significant horizontal acceleration places the end of the $\vec{a}_a$ vector close the boundary of allowable values. When this happens, noise or offset in either the

SLAM odometery or the accelerometer can frequently cause a no-solution condition. To get a solution, at least one of the parameters must change.

As $\vec{a}_a$ approaches the outer boundary of measurement consistency, $\vec{t}$ approaches $\vec{0}$. At the boundary, $\vec{t} = \vec{0}$, and Equation 2.12 simplifies to the vector projection of $\vec{a}_a$ onto $\vec{u}$,

$$\vec{w} = (\vec{a}_a \cdot \vec{u})\vec{u} \tag{2.13}$$

This is the shown by $\vec{a}_3$, in Figure 2.4 where $\vec{w}$, $\vec{v}$ and $\vec{a_a}$ make a right triangle, and $\vec{v} = \vec{g}$. If $\vec{a}_a$ extends into a disallowed region, the situation is similar, except $\vec{v}$ is longer than $\vec{g}$. We can get one approximate solution by using Equation 2.13. This approach essentially finds $\vec{w}$ with a larger value of $||\vec{g}||$. However, this is not the best option, since $||\vec{g}||$ is the one parameter that we know with the highest level of confidence. A better plan is to adjust one of the values that we have less confidence in. We can adjust either the direction or magnitude of $\vec{a}_a$, or the direction of $\vec{u}$. Since $\vec{u}$ is the least accurate measurement, this is the best parameter to change.



(a) Use larger value for $\vec{g}$ (b) Change angle of $\vec{u}s$

Fig. 2.8. It's easiest to just allow $\vec{g}$ to take on a larger value, but it's probably more accurate to change the angle of $\vec{u}$ and $\vec{w}$.

If $\vec{u}$ is rotated far enough towards $\vec{a}_a$, $||\vec{v}||$ will be equal to $||\vec{g}||$. This creates a new right triangle, where

$$||\vec{w_1}|| = \left(\sqrt{||\vec{a}||^2 - ||\vec{v_1}||^2}\right) \tag{2.14}$$

To find $\vec{w_1}$, we will use Rodrigues' rotation formula to rotate $\vec{a}_a$ into the direction of $\vec{w_1}$, and then scale the rotated vector to the known length of $\vec{w_1}$. It may seem strange that we are rotating $\vec{a}_a$ and not $\vec{u}$ or $\vec{w_0}$, since $\vec{u}$ is the parameter that is getting changed. The triangle diagrams in Figure 2.8 are two dimensional, but they are oriented arbitrarily in three dimensions. Using the known angle between $\vec{a}_a$ and $\vec{w_1}$, we can rotate $\vec{a}_a$ through our 2D plane in 3D space to get a vector in the vector space of $\vec{w_1}$. We obtain θ with

$$\theta = atan\left(\frac{||\vec{v_1}||}{||\vec{a}||}\right) \tag{2.15}$$

The rotation is then

$$\vec{l} = acos(\theta) + (k \times \vec{a}_a)sin(\theta) + k(k \cdot a)(1 - cos(\theta)) \tag{2.16}$$

where

$$\vec{k} = \frac{\vec{u} \times \vec{a}_a}{||\vec{u} \times \vec{a}_a||} \tag{2.17}$$

Because the rotation is entirely on the plane, $(k \cdot a) = \vec{0}$, and Equation 2.16 simplifies to

$$\vec{l} = acos(\theta) + (k \times \vec{a}_a^s)sin(\theta) \tag{2.18}$$

Finally, we get $\vec{w_1}$ by multiplying by the the ratio of the magnitudes of $\vec{w_1}$ and $\vec{a}_a$,

$$\vec{w_1} = \vec{l}\left(\frac{||\vec{w_1}||}{||\vec{a}_a||}\right) \tag{2.19}$$

2.4 Virtual IMU

Until this point, the SLAM camera and the accelerometer have been treated as if they physically occupied the same location. Obviously, this is impossible in practice,

so they must instead be rigidly attached to the same rigid body. For purely translational movement, sensor separation is not an issue, however, rotational movement can cause significantly different local acceleration. Obviously, it is best to locate the camera and the sensor as closely as possible, but in practice, the displacement may still be too large to ignore, especially for the kind rapid, unconstrained motion that the proposed scaling method is best suited.

Assuming the measurements have already been rotated into alignment, we can use the angular rate from a gyroscope mounted anywhere on that body to determine what a virtual accelerometer would read if it were mounted where the camera is.

$$\vec{a}_{vimu} = \vec{a}(\vec{\alpha} \times \vec{r}) + (\vec{\omega} \times (\vec{\omega} \times \vec{r})) \tag{2.20}$$

where $\vec{\alpha}$ is rotational acceleration, $\vec{\omega}$ is rotational rate, and $\vec{r}$ is the displacement vector between the accelerometer and the virtual accelerometer [14].

During implementation, the accelerometer was mounted only a few centimeters from the camera, but the measurement difference was relatively large with any significant rotation. Oddly, we were only able to find one other example of a rotation compensated VIMU being used in related literature [15], and they were similarly unable to find other published examples. Possibly the most broadly important takeaway from this thesis is that current research seems to often neglect for accelerometer displacement. It is relatively easy to compensate for, and should at least be considered in any application involving sensor fusion. The biggest drawback to doing this is that the non-linearity can cause problems with the covariance matrix for the sensor, and derivative of the gyroscope will require smoothing which will reduce the bandwidth of resulting signal.

2.5 Digital Filtering

The methods described here require the derivatives of two discrete signals; $\vec{\alpha}$, the first derivative of the gyroscope's angular rate signal, $\vec{\omega}$, and $\vec{a_s}$, the second derivative of the SLAM odometery position vector, $\vec{p}$. Numerical differentiation is

difficult, because differentiation tends to strongly amplify any high frequency noise in the signal. Strong filtering may be needed to remove this noise, which can seriously degrade the bandwidth of the signal, and require significant buffering.

The simplest method for getting a derivative of a discrete signal is to use a finite difference between two points. A backwards finite difference, for example, is simply the slope between the newest data point and the one before it

$$f'(n) \approx (f(n) - f(n-1))f_s \tag{2.21}$$

where f_s is the sampling frequency. A second order finite difference can be obtained with at least three samples,

$$f''(n) \approx (f(n+1) - 2f(n) + f(n-1))f_s^2 \tag{2.22}$$

Both of these equations amplified sensor and odometery noise to unusable levels. In order to adequately reduce this noise with a low pass filter, it would have been necessary to set such a low cutoff frequency that hand held linear acceleration would have been largely attenuated as well. Fortunately, better options exist. In 1964, a filter which came to be known as the Savitksy-Golay filter was proposed [16]. It turns out that in many cases, a least squares fit of a polynomial to a data signal can outperform a low pass filter in eliminating noise from a discrete signal. However, the high computational complexity of least squares fitting makes it undesirable for large window sizes. Savitksy and Golay found that coefficients could be found for a convolution filter that found the exact value of a least squares fit at the center of the filter. Even better, the coefficients can be computed instead to give the nth derivative of the polynomial fit. And, because a convolution filter is a linear combination of the input signal, the covariance matrix of the result is easily computed.

At several points, we also found it necessary to use low pass filters to match the bandwidth of the accelerometer and SLAM odometer acceleration signals. This is important, as higher frequency components in one signal will show up as noise in the scale estimate. The exact configuration of these filters is dependent on the

implementation, and will be a function of the data rates and noise of the sensors used. In order to make it easier track measurement covariance, we used FIR low pass filters, but this may or may not be the best approach for a particular implementation. The longer the filters, the more delay is introduced into the output, which is not desirable.

2.6 Scale Variance

When possible, it is desirable to know about the uncertainty of signals derived from sensors. The scaling method that has been described will naturally work better under some conditions than others. In order to have a sufficiently robust system, it may be necessary to use other sensors and other scaling methods. If the uncertainties associated with these estimates are known, we can elegantly determine the best compromise between conflicting values with common sensor fusion techniques such as an Extended Kalman Filter. Knowing the uncertainty of the scale allows for a more complete idea of the uncertainty of scaled SLAM odometery data. As has been mentioned before, the scaling method proposed in the previous sections will not produce a good estimate unless the system has sufficient linear acceleration to work with. If we can track the uncertainty of the scale, we can stabilize the result with an inverse-variance weighting of several estimates.

This section will describe a linearized estimate for scale variance and based on the co-variance matrices associated with the signal inputs from the accelerometer, gyroscope and SLAM odometery. The covariance matrices will be computed with linear functions which are approximated for each new set of function inputs. It has to be said that this approach is not entirely successful, as some of the vector equations linearize poorly, especially the vector norm. However, it is sufficient for inverse-variance weighting the scale, which allows the output to ignore highly uncertain estimates.

2.6.1 Covariance Matrix of Linear Combination of Random Vectors

To compute the covariance matrix associated with the linearized equations, we use the general form for the covariance matrix of a linear combination of random vectors. We will develop this form from the basic properties of random vectors. Let $\vec{y}$ represent a linear combination of random vectors $\vec{x}_1 ... \vec{x}_n$, with matrix coefficients $A_1 ... A_n$,

$$\vec{y} = A_1\vec{x}_1 + A_2\vec{x}_2 + ... + A_n\vec{x}_n \tag{2.23}$$

The covariance matrix of $\vec{y}$ is therefore

$$\Sigma_y = Var(\vec{y}) = Var(A_1\vec{x}_1 + A_2\vec{x}_2 + ... + A_n\vec{x}_n) \tag{2.24}$$

But,

$$Var(\vec{y}) = Cov(\vec{y}, \vec{y}) \tag{2.25}$$

Which can be written

$$\Sigma_y = Cov(A_1\vec{x}_1 + A_2\vec{x}_2 + ... + A_n\vec{x}_n, A_1\vec{x}_1 + A_2\vec{x}_2 + ... + A_n\vec{x}_n) \tag{2.26}$$

and because covariance is distributive,

$$\Sigma_y = \sum_{i=1}^{n} Cov(A_i\vec{x}_i, A_i\vec{x}_i) + \sum_{i \neq j} Cov(A_i\vec{x}_i, A_j\vec{x}_j) \tag{2.27}$$

Using the following two properties of covariance matrices,

$$Var(A\vec{y}) = A\Sigma_y A^T \tag{2.28}$$

and

$$Cov(A\vec{y}, B\vec{x}) = ACov(\vec{y}, \vec{x})B^T \tag{2.29}$$

We can rewrite Equation 2.27 as,

$$\Sigma_y = \sum_{i=1}^{n} A_i\Sigma_{x_i}A_i^T + \sum_{i \neq j} A_i Cov(\vec{x}_i, \vec{x}_j)A_j^T \tag{2.30}$$

2.6.2 VIMU Covariance Matrix

Because convolution filters are implementation specific and straightforward linear operations, dealing with their covariance matrices will not be addressed here. We will begin with the VIMU, finding the linearized covariance matrix of the virtual accelerometer as a function of the accelerometer's and gyroscope's covariance matrices. $\vec{a}_a$ has been used to denote acceleration at the point of the camera, so we will use $\vec{a}_d$ to represent the acceleration at the displaced accelerometer. $\Sigma_{\vec{a}_d}$ will represent the covariance matrix associated with the accelerometer's covariance matrix. Starting with the VIMU equation,

$$\vec{a}_{vimu} = f(\vec{a}_d, \vec{\alpha}, \vec{\omega}) = \vec{a}(\vec{\alpha} \times \vec{r}) + (\vec{\omega} \times (\vec{\omega} \times \vec{r})) \tag{2.31}$$

we get the form of the linearized approximation,

$$\vec{a}_{vimu} \approx f(\vec{a}_{d0}, \vec{\alpha}_0, \vec{\omega}_0) + A_a\vec{a}_d + B_a\vec{\alpha} + C_a\vec{\omega} \tag{2.32}$$

The coefficients A_a, B_a and C_a are the Jacobians of $f(\vec{a}_d, \vec{\alpha}, \vec{\omega})$ with respect to each variable evaluated at the function input values, $\vec{a}_{d0}, \vec{\alpha}_0, \vec{\omega}_0$, resulting in,

$$A_a = J_f(\vec{a}_d) = I \tag{2.33}$$

and

$$B_a = J_f(\vec{\alpha}) = \begin{bmatrix} 0 & r_3 & -r_2 \\ -r_3 & 0 & r_1 \\ r_2 & -r_1 & 0 \end{bmatrix} \tag{2.34}$$

and

$$C_a = J_f(\vec{\omega}) = \begin{bmatrix} \omega_2 r_2 + \omega_3 r_3 & \omega_1 r_2 - 2\omega_2 r_1 & \omega_1 r_3 - 2\omega_3 r_1 \\ \omega_2 r_1 - 2\omega_1 r_2 & \omega_3 r_3 + \omega_1 r_1 & \omega_2 r_3 - 2\omega_3 r_2 \\ \omega_3 r_1 - 2\omega_1 r_3 & \omega_3 r_2 - 2\omega_2 r_3 & \omega_1 r_1 + \omega_2 r_2 \end{bmatrix} \tag{2.35}$$

Now that we have approximated the function as a linear combination of the inputs, we can use the general form for the variance of a linear combination of random variables to find

$$\Sigma_s = \Sigma_{a_d} + B_a \Sigma_\alpha {B_a}^T + C_a \Sigma_\omega {C_a}^T + B_a Cov(\alpha, \omega) {C_a}^T + C_a Cov(\omega, \alpha) {B_a}^T \quad (2.36)$$

where $Cov(\alpha, \omega)$ and $Cov(\omega, \alpha)$ is determined by the derivative operation used in the implementation.

2.6.3 Standard Scale Variance

The first covariance matrix that needs to be calculated is Σ_u, the unit vector of $\vec{a}_s$. This linearizes particularly poorly, because the uncertainty of $\vec{u}$ lies on the surface of a sphere of magnitude 1. However, this linearizion extends the uncertainty into a 3D cloud around $\vec{u}$. In practice, this seems to vastly overestimate variance, and tends to cause the other covariance matrices which depends to blow up. From

$$\vec{u} = \frac{\vec{a}_s}{\|\vec{a}_s\|} \quad (2.8)$$

the linear approximation takes the form

$$\vec{u} \approx f(\vec{a}_{s0}) + A_u \vec{a}_s \quad (2.37)$$

Where the Jacobian A_u is evaluated at $\vec{a}_{s0}$,

$$A_u = J_f(\vec{a}_s)|_{\vec{a}_{s0}} = \left[\begin{array}{ccc} \frac{||\vec{a}_s||^2 - a_{s1}}{||\vec{a}_s||^3} & \frac{-a_{s1}a_{s2}}{||\vec{a}_s||^3} & \frac{-a_{s1}a_{s3}}{||\vec{a}_s||^3} \\ \frac{-a_{s1}a_{s2}}{||\vec{a}_s||^3} & \frac{||\vec{a}_s||^2 - a_{s2}}{||\vec{a}_s||^3} & \frac{-a_{s2}a_{s3}}{||\vec{a}_s||^3} \\ \frac{-a_{s1}a_{s3}}{||\vec{a}_s||^3} & \frac{-a_{s2}a_{s3}}{||\vec{a}_s||^3} & \frac{||\vec{a}_s||^2 - a_{s3}}{||\vec{a}_s||^3} \end{array} \right] \Bigg|_{\vec{a}_{s0}} \quad (2.38)$$

The linearized covariance matrix of $\vec{u}$ is therefore

$$\Sigma_u = A_u \Sigma_{a_s} {A_u}^T \quad (2.39)$$

Using this result, we can deal with Equation 2.10, the vector rejection of $\vec{a}_a$ onto $\vec{u}$,

$$\vec{v} := \vec{a}_a - (\vec{a}_a \cdot \vec{u})\vec{u} \quad (2.10)$$

and the linear approximation takes the form

$$\vec{v} \approx f(\vec{a}_{a0}, \vec{u}_0) + A_v \vec{a}_a + B_v \vec{u} \tag{2.40}$$

where Jacobians A_v and B_v are

$$A_v = J_f(\vec{a}_a)|_{\vec{u}_0} = \left[\begin{matrix} 1-u_1^2 & -u_1u_2 & -u_1u_3 \\ -u_1u_2 & 1-u_2^2 & -u_2u_3 \\ -u_1u_3 & -u_2u_3 & 1-u_3^2 \end{matrix}\right]\Bigg|_{\vec{u}_0} \tag{2.41}$$

and

$$\begin{aligned} B_v &= J_f(\vec{u})|_{\vec{a}_{a0},\vec{u}_0} \\ &= \left[\begin{matrix} -2a_{a1}u_1 - a_{a2}u_2 - a_{a3}u_3 & -a_{a2}u_1 & -a_{a3}u_1 \\ -a_{a1}u_2 & -a_{a1}u_1 - 2a_{a2}u_2 - a_{a3}u_3 & -a_{a3}u_2 \\ -a_{a1}u_3 & -a_{a2}u_3 & -a_{a1}u_1 - a_{a2}u_2 - 2a_{a3}u_3 \end{matrix}\right]\Bigg|_{\vec{a}_{a0},\vec{u}_0} \end{aligned} \tag{2.42}$$

Because $\vec{a}_a$ and $\vec{u}$ are independent, the expression for Σ_v simplifies to

$$\Sigma_v = A_v \Sigma_{a_a} {A_v}^T + B_v \Sigma_u {B_v}^T \tag{2.43}$$

Next we compute the Covariance Matrix for $\vec{t}$, found in Equation 2.11,

$$\vec{t} := \left(\sqrt{||\vec{g_n}|| - ||\vec{v}||}\right) \vec{u} \tag{2.11}$$

The linear approximation takes the form

$$\vec{t} \approx f(\vec{v}_0, \vec{u}_0) + A_t \vec{v} + B_t \vec{u} \tag{2.44}$$

Where the Jacobians A_t and B_t are

$$A_t = J_f(\vec{v})|_{\vec{v}_0,\vec{u}_0} = \left[\begin{matrix} \frac{-v_1u_1}{m} & \frac{-v_1u_2}{m} & \frac{-v_1u_3}{m} \\ \frac{-v_2u_1}{m} & \frac{-v_2u_2}{m} & \frac{-v_2u_3}{m} \\ \frac{-v_3u_1}{m} & \frac{-v_3u_2}{m} & \frac{-v_3u_3}{m} \end{matrix}\right]\Bigg|_{\vec{v}_0,\vec{u}_0,m=\sqrt{9.81^2-v_1^2-v_2^2-v_3^2}} \tag{2.45}$$

and

$$B_t = J_f(\vec{v})|_{\vec{v}_0,\vec{u}_0} = \left[\begin{matrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & m & m \end{matrix}\right]\Bigg|_{m=\sqrt{9.81^2-v_1^2-v_2^2-v_3^2}} \tag{2.46}$$

leading us to the expressions for Σ_t,

$$\Sigma_t = A_t \Sigma_v A_t^T + B_t \Sigma_u B_t^T + A_t Cov(\vec{v}, \vec{u}) B_t^T + B_t Cov(\vec{u}, \vec{v}) A_t^T \tag{2.47}$$

where

$$Cov(\vec{v}, \vec{u}) = Cov(A_v \vec{a}_a + B_v \vec{u}, A_u \vec{a}_s) = B_v Cov(\vec{u}, \vec{a}_s) A_u^T + 0 \tag{2.48}$$

Terms with covariance between independent variables have dropped out. By devloping the covariance expressions, we get

$$Cov(\vec{u}, \vec{a}_s) = Cov(A_u \vec{a}_s, \vec{a}_s) = A_u \Sigma_{a_s} \tag{2.49}$$

and using similar reasoning for $Cov(\vec{u}, \vec{v})$, we get the final result,

$$\Sigma_t = A_t \Sigma_v A_t^T + B_t \Sigma_u B_t^T + A_t B_v A_u \Sigma_{a_s} A_u^T B_t^T + B_t A_u \Sigma_{a_s} A_u^T B_v^T A_t^T \tag{2.50}$$

Next we obtain linear acceleration covariance, Σ_w. Recall Equation 2.12,

$$\vec{w} = (\vec{a}_a \cdot \vec{u})\vec{u} \pm \vec{t} \tag{2.12}$$

The linear approximation takes the form

$$\vec{w} \approx f(\vec{a}_a, \vec{u}_0, \vec{t_0}) + A_w \vec{a}_a + B_w \vec{u} + C_w \vec{t} \tag{2.51}$$

where the Jacobians A_w and B_w and C_w are

$$A_w = J_f(\vec{a}_a)|_{\vec{a}_{a0}, \vec{u}_0, \vec{t_0}} = \begin{bmatrix} u_1^2 & u_1 u_2 & u_1 u_3 \\ u_1 u_2 & u_2^2 & u_2 u_3 \\ u_1 u_3 & u_2 u_3 & u_3^2 \end{bmatrix} \Bigg|_{\vec{u}_0} \tag{2.52}$$

and

$$\begin{aligned} B_w &= J_f(\vec{u})|_{\vec{a}_{a0}, \vec{u}_0, \vec{t_0}} \\ &= \begin{bmatrix} 2a_{a1}u_1 + a_{a2}u_2 + a_{a3}u_3 & -a_{a2}u_1 & a_{a3}u_1 \\ a_{a1}u_2 & a_{a1}u_1 + 2a_{a2}u_2 + a_{a3}u_3 & a_{a3}u_2 \\ a_{a1}u_3 & a_{a2}u_3 & a_{a1}u_1 + a_{a2}u_2 + 2a_{a3}u_3 \end{bmatrix} \Bigg|_{\vec{a}_{a0}, \vec{u}_0} \end{aligned} \tag{2.53}$$

and

$$C_w = J_f(\vec{t}) = \pm I \tag{2.54}$$

Allowing us to solve for Σ_w,

$$\Sigma_w = A_w\Sigma_{a_a}A_w^T + B_w\Sigma_u B_w^T + C_w\Sigma_t C_w^T + A_w Cov(\vec{a}_a, \vec{u})C_w^T + B_w Cov(\vec{u}, \vec{t})C_w^T + C_w Cov(\vec{u}, \vec{a}_a)A_w^T + C_w Cov(\vec{t}, \vec{u})B_w^T \quad (2.55)$$

where terms with covariance between independent variables have dropped out. Devloping the covaraince terms,

$$Cov(\vec{a}_a, \vec{t}) = Cov(\vec{a}_a, A_t\vec{v} + B_t\vec{u}) = Cov(\vec{a}_a, \vec{v})A_t^T + 0 \quad (2.56)$$

and

$$Cov(\vec{a}_a, \vec{v}) = \Sigma_{a_a}A_v^T + 0 \quad (2.57)$$

and

$$Cov(\vec{a}_a, \vec{t}) = \Sigma_{a_a}A_v^T A_t^T \quad (2.58)$$

and

$$Cov(\vec{u}, \vec{t}) = Cov(\vec{u}, A_t\vec{v} + B_t\vec{u}) = Cov(\vec{u}, \vec{v})A_t^T + \Sigma_u B_t^T \quad (2.59)$$

and

$$Cov(\vec{u}, \vec{v}) = B_v A_u \Sigma_w A_u^T \quad (2.60)$$

and

$$Cov(\vec{u}, \vec{t}) = B_v A_u \Sigma_w A_u^T A_t^T + \Sigma_u B_t^T \quad (2.61)$$

Using similar reasoning for $Cov(\vec{t}, \vec{a}_a)$ and $Cov(\vec{t}, \vec{u})$, we we obtain

$$\Sigma_w = A_w\Sigma_{a_a}A_w^T + B_w\Sigma_u B_w^T + C_w\Sigma_t C_w^T + A_w\Sigma_{a_a}A_v^T A_t^T C_w^T + B_w(B_v A_u \Sigma_w A_u^T A_t^T + \Sigma_u B_t^T)C_w^T + C_w A_t A_v \Sigma_{a_a} A_w^T + C_w(A_t A_u \Sigma_w A_u^T B_v^T + B_t\Sigma_u)B_w^T \quad (2.62)$$

To find the scale co-variance, we start with

$$s = \frac{||\vec{w}||}{||\vec{a}_s||} \quad (2.63)$$

This time the linear approximation is a scalar function,

$$s = f(\vec{w}_0, \vec{a}_{s0}) + \vec{a}_{scale}\vec{w} + \vec{b}_{scale}\vec{a}_s \quad (2.64)$$

where $\vec{a}_{scale}$ and $\vec{b}_{scale}$ are

$$\vec{a}_{scale} = \frac{\partial}{\partial \vec{w}} \left(\frac{||\vec{w}||}{||\vec{a}_s||} \right) \Big|_{\vec{a}_{a0}, \vec{w}_0} = \frac{\vec{w}}{||\vec{w}||||\vec{a}_s||} \Big|_{\vec{a}_{s0}, \vec{w}_0} \tag{2.65}$$

and

$$\vec{b}_{scale} = \frac{\partial}{\partial \vec{a}_s} \left(\frac{||\vec{w}||}{||\vec{a}_s||} \right) \Big|_{\vec{a}_{s0}, \vec{w}_0} = \frac{-||\vec{w}||\vec{w}}{||\vec{a}_s||^3} \Big|_{\vec{a}_{s0}, \vec{w}_0} \tag{2.66}$$

Next, we get the expression for σ_s^2, the linearized variance of s,

$$\sigma_s^2 = \vec{a}_{scale} \Sigma_w \vec{a}_{scale}^T + \vec{b}_{scale} \Sigma_{a_s} \vec{b}_{scale}^T + 2\vec{a}_{scale} Cov(\vec{w}, \vec{a}_s) \vec{b}_{scale}^T \tag{2.67}$$

which can be expanded to

$$\sigma_s^2 = \vec{a}_{scale} \Sigma_w \vec{a}_{scale}^T + \vec{b}_{scale} \Sigma_{a_s} \vec{b}_{scale}^T + 2\vec{a}_{scale} (B_w A_u \Sigma_{a_s} + C_w (A_t B_v A_u \Sigma_{a_a} + B_t A_u \Sigma_{a_s})) \vec{b}_{scale}^T \tag{2.68}$$

2.6.4 Alternate Scale Variance

Estimating scale variance in the case of the alternate scale calculation method is less straightforward. Since $\vec{a_s}$ is changed due to accommodate other information, its covariance matrix is less valid. However, for the sake of simplicity, we will use the Σ_u for Σ_{u_1}. From there we can use the Pythagorean theorem to get an expression for $\vec{w}$ in terms of $\vec{a}$, $\vec{g_n}$, and $\vec{U}$. We will not treat $\vec{g_n}$ as a random variable, as its magnitude is not random. This gives us an expression for $\vec{w}$ that is highly analogous for the expression for $\vec{t}$,

$$\vec{t} = \left(\sqrt{||\vec{g_n}|| - ||\vec{v}||} \right) \vec{u} \rightarrow \vec{w} = \left(\sqrt{||\vec{a}_a|| - ||\vec{g_n}||} \right) \vec{u} \tag{2.69}$$

The linear approximation then becomes

$$\vec{w} \approx f(\vec{a_{a0}}, \vec{u}_0) + A_t \vec{a_a} + B_t \vec{u} \tag{2.70}$$

Where the Jacobians A_w and B_w are

$$A_w = J_f(\vec{a_a})|_{\vec{a_{a0}}, \vec{u}_0} = \begin{bmatrix} \frac{a_a 1 u_1}{m} & \frac{a_a 1 u_2}{m} & \frac{a_a 1 u_3}{m} \\ \frac{a_a 2 u_1}{m} & \frac{a_a 2 u_2}{m} & \frac{a_a 2 u_3}{m} \\ \frac{a_a 3 u_1}{m} & \frac{a_a 3 u_2}{m} & \frac{a_a 3 u_3}{m} \end{bmatrix} \Bigg|_{\vec{a_{a0}}, \vec{u}_0^s, m = \sqrt{v_1^2 + v_2^2 + v_3^2 - 9.81^2}} \tag{2.71}$$

and

$$B_w = J_f(\vec{u})|_{\vec{a_a}_0,\vec{u}_0} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & m & m \end{bmatrix}\Bigg|_{m=\sqrt{v_1^2+v_2^2+v_3^2-9.81^2}} \tag{2.72}$$

Because $\vec{a_a}$ and $\vec{u}$ are independent, Σ_w doesn't have the covariance terms of Σ_t,

$$\Sigma_w = A_w \Sigma_{\vec{v}} A_w^T + B_w \Sigma_{\vec{u}} B_w^T \tag{2.73}$$

This simplifies the expression for the variance of the scale,

$$\sigma_s^2 = \vec{a}_{scale} \Sigma_w \vec{a}_{scale}^T + \vec{b}_{scale} \Sigma_{a_a} \vec{b}_{scale}^T + 2\vec{a}_{scale}(B_w A_u \Sigma_{a_a})\vec{b}_{scale}^T \tag{2.74}$$

3. IMPLEMENTATION

This chapter will discuss the detailed implementation of the scaling and gravity detection system outlined in the Methods chapter. This will consist of four major topics; the hardware, the monocular simultaneous localization and mapping (SLAM) algorithm, the software interface, and the scaling code itself.

3.1 Hardware

There are three main hardware components in this system. The software primarily runs on a standard laptop computer running Ubuntu 14.04, and there are two USB hardware peripherals for sensor input, the inertial measurement unit (IMU) and the camera. The particular laptop isn't significant, so the detailed discussion will be confined to the IMU and camera.

3.1.1 IMU

The IMU was implemented on an ARM Mbed capable development board, because at the beginning of the project, the simplicity and flexibility of the Mbed platform was considered important. Mbed is a software platform which enables simple cross platoform embeded development across a wide range of ARM Cortex M based microcontrollers in C++. Mbed has reasonably good peripheral driver support, and greatly streamlines device configuration compared to typical device tool-chains.

The particular development board was chosen for its support of the rosserial-mbed library which allows Mbed devices to communicate with Robot Operating System (ROS), a software interface discussed further in Section 3.3. Later it became desirable to change the IMU for a device with factory calibration and measurement covariance

matrix estimates, however it was discovered that devices that supported covariance matrix estimation and had ROS support had all been discontinued, or else were impractically expensive. Therefore, the Mbed IMU was used in the final configuration.

NUCLEO-F401RE The NUCLEO-F401RE is an ST development board for the STM32F401RE microcontroller, which is based on the ARM Cortex M4F processor core [17]. The development board itself has an Arduino compatible header layout, and a built in ST-LINK programmer debugger chip that also implements a USB serial port.

X-NUCLEO-IKS01A1 The IKS01A1 is a sensor evaluation daughter board designed to interface with NUCLEO development boards [18]. It has a number of sensor integrated circuits including a 3-axis accelerometer / gyroscope, a 3-axis magnetometer, a barometer, and a humidity / temperature sensor. This project only requires the LSM6DS0 accelerometer / gyroscope [19].

The IKS01A1 has an Mbed compatible driver provided by ST which allowed for relatively painless configuration of the parts. However, it was discovered that the maximum data rate possible for simultaneous use of the accelerometer and gyroscope was 160Hz. Either the chip or the bus limits the data rate to less than either one is capable of alone. Although higher rates may be desirable for other applications, this was sufficient for working with monocular SLAM.

3.1.2 Camera

The camera is an IDS UI-1221LE-M-GL monochrome USB camera [20], chosen primarily because it was the same camera used by the group that developed our monocular SLAM system, LSD-SLAM. It was paired with an S-Mount 2.6mm wide angle lens. It only has a resolution of 752x480, but this is what LSD-SLAM was intended to work with. Most importantly, it is capable of high frame rates up to 87.2hz, and has a global shutter which eliminates shutter movement distortion.

A major issue that developed with this camera was that it did not to perform well above 40Hz on our test laptop. This was tracked down to an issue related to the camera's Linux driver and the rest states for the laptop's specific Intel i7 CPU, but we weren't able to resolve the problem. We did confirm that the issue did not exist another another computer, but LSD-SLAM crashes without error on that system, and we were also unable to resolve that issue. In the end, we were forced to make due with a 40hz frame rate.

3.2 Monocular SLAM

We chose LSD-SLAM, or Large Scale Direct SLAM for the test implementation of our scaling system [5]. LSD-SLAM is a very interesting monocular SLAM recently proposed which works directly on image pixels instead of image features, resulting in a far more detailed depth map. This method is appealing because it is can do this in real time by only comparing a subset of image pixels based on their local intensity gradient and stereo separation.

LSD-SLAM does stereo comparisons between new frames and irregularly generated keyframes, which are generated whenever the camera moves or pans too far from the last keyframe. The depth map is a point cloud of per-pixel depth estimates which are built against each keyframe. When a new keyframe is generated, the old one is added to a graph, where all the depth maps are aligned to form a global map. Each time a keyframe is added, LSD-SLAM uses large scale loop detection to find overlaps between the new frame's depth maps and the depth maps of other frames in the graph. If connections are found, the maps is re-optimized to align the keyframes and reduce drift.

The position and orientation of each new frame is estimated relative to the old keyframe via direct image alignment. When the relative poses of the frame are known, the depth map is generated from stereo comparisons on the subset of pixels which are

expected to yield accurate results. If a new keyframe is being generated, scale is then also estimated by adding the depth map to the image alignment optimization term.

3.3 Robot Operating System

The open source LSD-SLAM codebase is designed to work with ROS, or Robot Operating System. ROS is an open source platform used primarily by robotics researchers, which facilitates and abstracts communication across hardware devices and processes, and allows modular development of collaborative open source robotic systems.

ROS primarily runs on Linux based PCs, and communicates with peripherals via. device libraries. ROS programs, or nodes, can communicate with a roscore process which resides on a remote or local PC. The roscore process coordinates nodes and communication between nodes and device drivers, abstracting network and driver interfaces from the nodes. Nodes communicate by sending messages to roscore, which are published as topics, which other nodes can subscribe to.

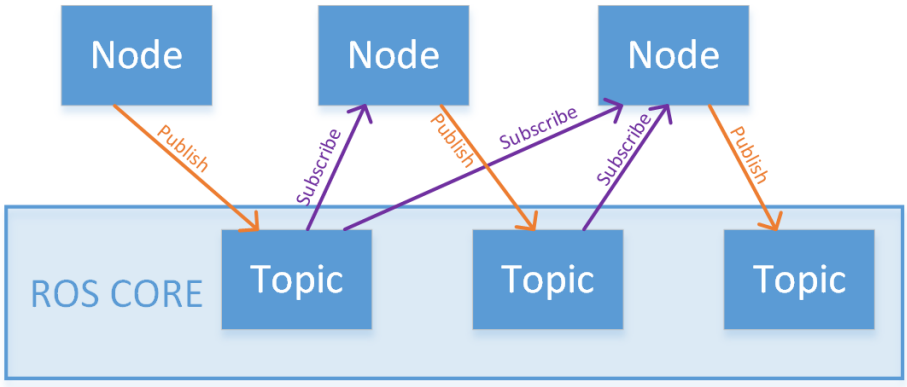


Fig. 3.1. ROS node-topic communication diagram.

The scale code was implemented on ROS Indigo, which is an older release of ROS, in order to ensure compatibility with LSD-SLAM.

3.4 Scale Code Design

All code was written in C++ in ROS nodes. The scaling system consists of four ROS nodes, one of which runs on the NUCLEO-F401RE Mbed device, and three which run on the laptop. This modular approach makes development and troubleshooting slightly easier. The first node, the Mbed IMU, publishes raw accelerometer and gyroscope data from the development board. The VIMU node scales those raw values, filters them, and handles the VIMU calculation. The Odometry Transformation node subscribes to LSD-SLAM topics, applies the second derivative to the SLAM odometery, and transforms the SLAM odometry into the body frame. Finally, the Scaler node does the calculations and filtering necessary to find the scale and gravity estimate.

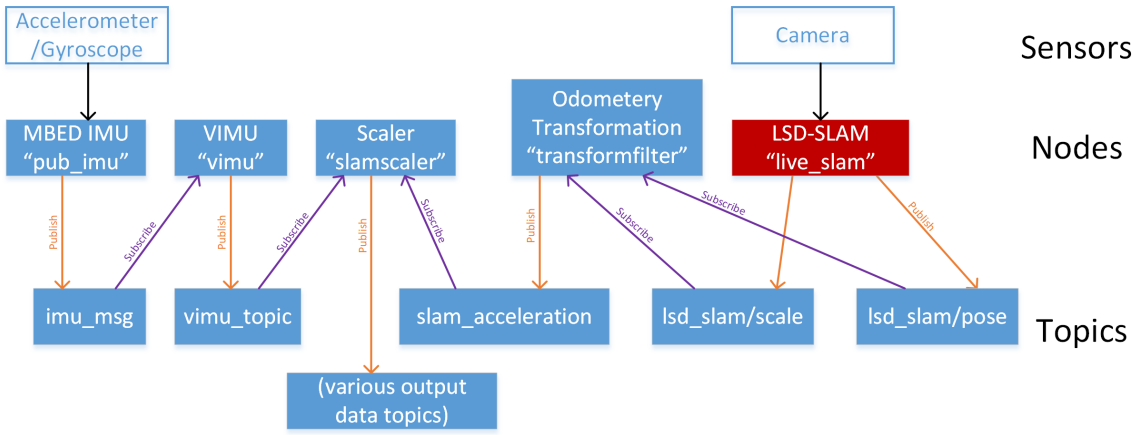


Fig. 3.2. High Level node block diagram.

3.4.1 MBED IMU Node

The Mbed IMU node runs on the NUCLEO-F401RE development board. Functionally, it simply calls the accelerometer/gyroscope's device driver to request samples at the highest possible rate. Experimentally, this was determined to be 160Hz, though

even at this rate, samples are often repeated, creating stair-step noise in the signal. At first it seemed to be quantization error, but the steps occurred on steep slopes that were otherwise constant and smooth. These issues were corrected with FIR filters in the VIMU node.

In order to get the information from the device to the computer, the rosserial_mbed library was used, which connects to a rosserial python node on the host computer and allows the Mbed device to communicate with the roscore master process and publish topics.

3.4.2 VIMU Node

The VIMU filters the raw internal signals, applies a limited calibration, and processes the VIMU calculation from section 2.4. It also produces a linearized estimate for the covariance matrix of the final linear acceleration vector.

The accelerometer was crudely calibrated by approximately pointing each axis toward the ground and offsetting the value to make it approximately $9.81 m/s^2$. The gyroscope was calibrated by laying it on the table and offsetting the rates so they read zero. These offsets are applied to the signals and then multiplied by a factor to put them in ROS standard units of m/s^2 and radians per second.

After this both signals are put into buffers and filtered with an FIR low pass filter to remove high frequency noise. The filtered signals are put into another buffer so that a first derivative Savitsky-Golay filter can be computed for the angular rate signal, and the other signals can be be synchronized with the result. Finally the VIMU calculation is performed and the results are published to the "vimu" topic.

Because no IMU with ROS support that published a covariance estimate of its measurements was available for ROS at the time of writing, a constant covariance matrix was estimated and hard coded instead. However this was done at the beginning of the process, so that if a covariance matrix estimate from the device were available, it could immediately be implemented. The gyroscope's and accelerometer's covariance

matrices are buffered with every step of their respective signals, and are updated to accommodate the affect of the filters. The final vimu vector's covariance matrix is published in a topic with the vimu vector.

Throughout the code, convolution filters have their corresponding signal covariance matrices updated with the following equation:

$$\Sigma_{filtered} = \sum_{i=1}^{n} \vec{\zeta_i} \Sigma_i \vec{\zeta_i^T} \tag{3.1}$$

Where Σ_i is the covariance matrix for each signal value in the filter, and $\vec{\zeta_i}$ is a vector of the filter coefficient for that value. All signals are assumed to comprised of independent random variables.

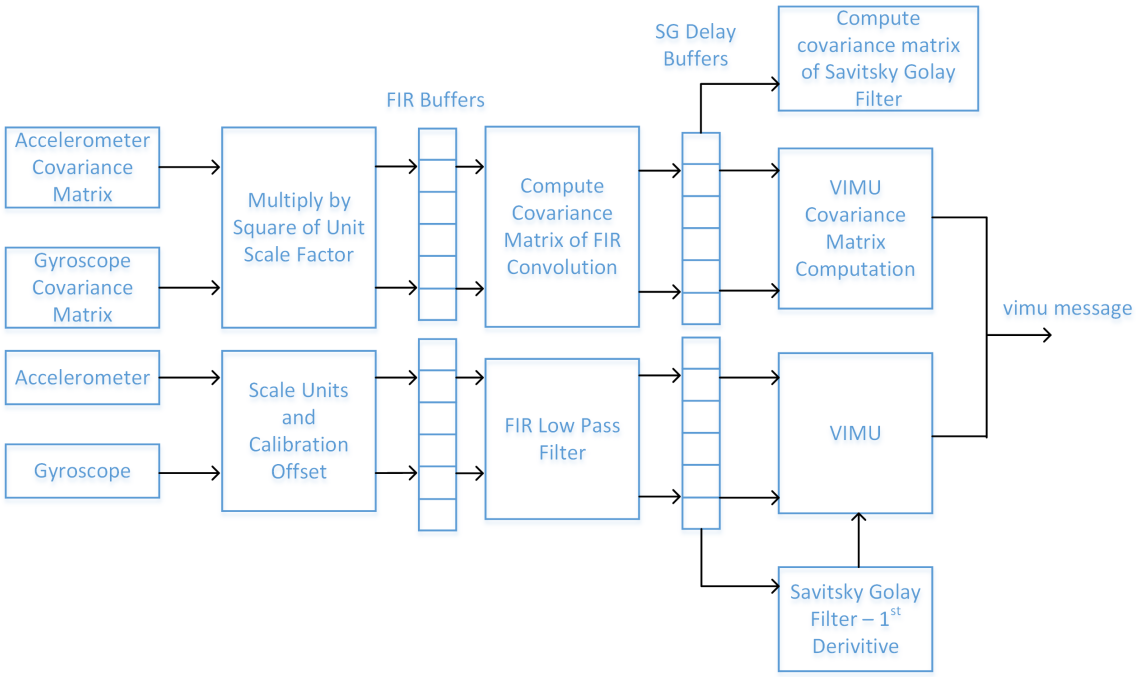


Fig. 3.3. VIMU node block diagram.

3.4.3 Odometry Transformation Node

This node handles the LSD-SLAM odometry and produces the linear acceleration estimate that is used by the main scaler code. It deals with reference frames differently to how they were proposed in the Methods chapter. Instead of transforming the VIMU into the SLAM frame, the SLAM acceleration is transformed into the body frame. It was realized late in the code development that this was an inferior approach for reasons that were discussed in 2.3. The main issue with the body frame is that it is not a single frame but a series of frames rotated from one another. Because convolution operations such as FIR or Savitsky-Golay filters assume that all elements are in the same reference frame, operating in the body frame does introduce some element of error. However, because the filter lengths used are small, and the main goal is to reduce high frequency noise in the signal, the error it introduces is not large enough to significantly affect the result.

As is the case with the IMU, LSD-SLAM does not provide an odometry covariance matrix estimate, and its authors believe its internal covariance estimate to be highly inaccurate. Therefore, a fixed covariance matrix was used in lieu of live data, again at the beginning of the calculation to more accurately simulate a situation with a proper covariance matrix estimate. However, because larger scale scenes must necessarily have more noise, LSD-SLAM was modified to publish the internal scale of the current keyframe. This scale was used to modify the fixed covariance matrix in order to compensate for the relative change in noise compared to the initial LSD-SLAM internal scale value. Obviously it is far better to have a realistic covariance estimate from the SLAM algorithm itself, but this enhances the relative variance estimate that is needed to down-weight low confidence estimates of the final metric scale.

Before doing anything else with the SLAM odometery signal, the coordinate basis vectors are swapped to conform to ROS guidelines. After this, the signal and covariance matrix is buffered for the Savtisky-Golay second derivative filter, which is then

performed and the covariance matrix is updated. The Savtisky-Golay filter used has a length of nine and a polynomial order of three. This was found to be a reasonable compromise between preserving hand-held motion and smoothing higher frequency noise for the 40Hz camera frame rate that was used.

The linear acceleration signal and its covariance matrix are passed to a library called "pose_cov_ops" which uses the pose orientation to rotate the acceleration into the body frame and update the covariance matrix. The result is then published to the slam_acceleration topic.

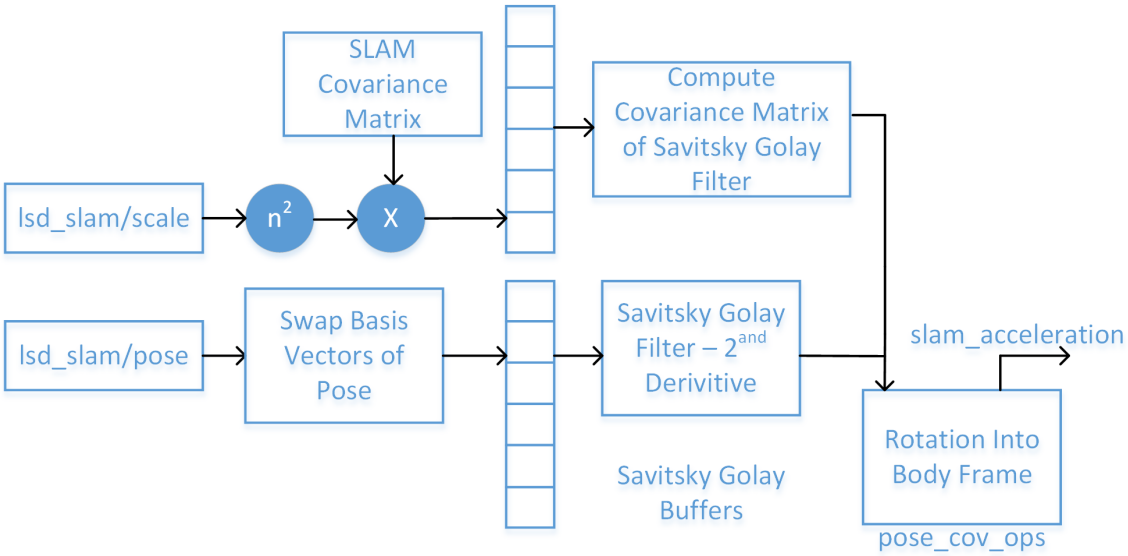


Fig. 3.4. Odometry Transform node block diagram.

3.4.4 Scaler Node

This is the final node and is responsible for making the final calculations to determine scale and the gravity vector. The SLAM derived linear acceleration signal and the VIMU signal have different bandwidths and sample rates. In the scale estimation, aliasing artifacts or even dissimilar bandwidths will manifest as noise in the final scale

estimate. In order to fix this, the VIMU was passed through an additional FIR low pass filter to reduce its bandwidth to that of the SLAM signal. The Nyquist rate of the SLAM signal was 20Hz in our implementation, but the Savitsky-Golay filter reduced its total bandwidth to roughly 10Hz. Matching the SLAM signal's bandwidth is imprecise, because while the Savitsky-Golay filter is roughly a low pass filter, by design it has extreme ripple in the stop band [21]. Filter choice should be carefully tweaked for the particular implementation.

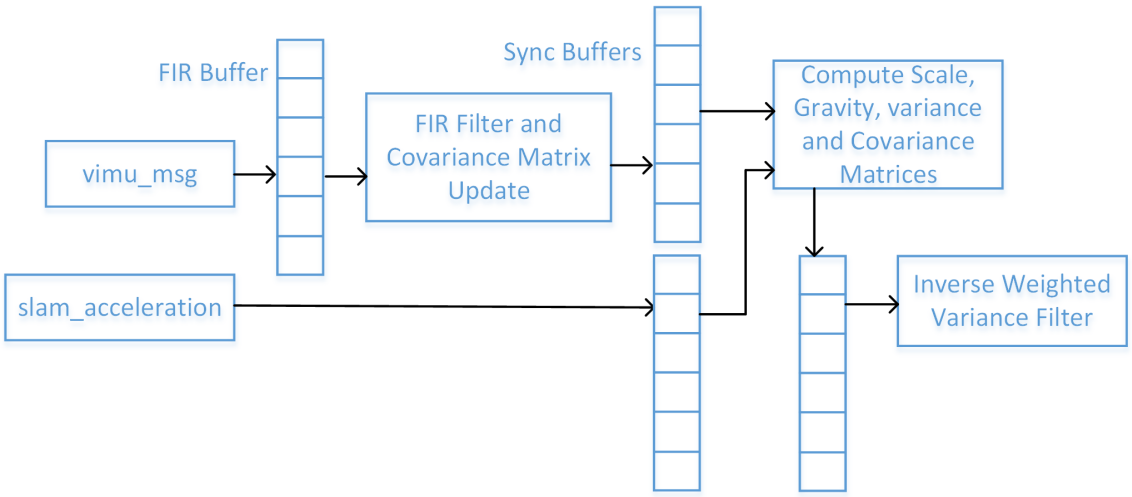


Fig. 3.5. Scaler node block diagram.

The scale calculation can only be done at the rate of the slowest signal. In this case, that will always be the SLAM signal. In order to synchronize the two signals, both the VIMU and SLAM signals are buffered. The timestamp for each SLAM vector is used to find the closest time adjacent VIMU vectors, which are then linearly interpolated to the time-stamp of the SLAM vector. After this, older values in both buffers are deleted. The covariance matrices are also buffered and computed to compensate for these operations.

At this point the synchronized, filtered SLAM linear acceleration and VIMU vectors are used to compute the gravity and scale. The linearized covariance matrices

are computed, and both results are put into a one second buffer. A moving inverse variance weighted average stabilizes the result and downweighs scale estimates with an insufficient signal to noise ratio. The same process can be used for the gravity estimate.

$$\hat{scale} = \frac{\sum_i s_i/\sigma_{si}^2}{\sum_i 1/\sigma_{si}^2} \tag{3.2}$$

4. DISCUSSION OF RESULTS

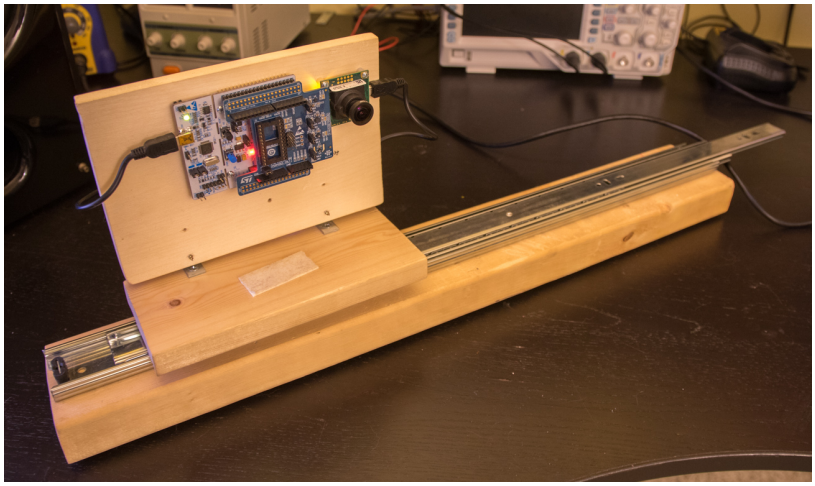
The scaling method proposed in this thesis is promising, and can produce good results given low enough noise from SLAM odometery. The proof of concept implementation has a number of limitations which make the scale somewhat unstable, but with a high enough signal to noise ratio, it can be coaxed into providing some good examples of what this method is potentially capable of. The following chapter will show and discuss tests of the implementation system's ability to measure scale in various scenarios. It will also test the performance of its linear acceleration estimate which will provide insight into the quality of the gravity vector estimate, which we did not have the equipment to test directly. There is also a demonstration of the effectiveness and significance the VIMU.

As has been mentioned previously, the covariance estimate was too unrealistic to directly use with sensor fusion techniques, but it did prove extremely valuable for downweighing the effect of very large, unreliable estimates which made the inverse variance weighted scale far more stable than a simple moving average filter could have.

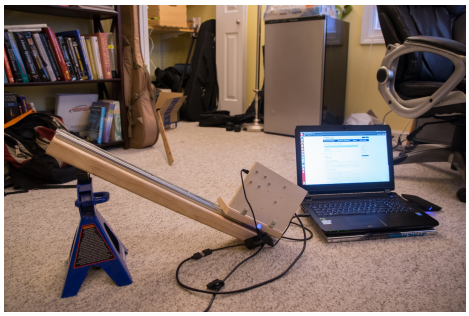
4.1 Test Equipment Configuration

In addition to the equipment described in the Implementation chapter, we have put together a simple device to explore the accuracy of the scale estimate. In order to test whether the scale factor produced is a realistic, we configured the software to produce a real time scaled graph of SLAM odometery position. We attached the camera to a drawer pull mounted to a two by four wood beam. By hand sliding the camera along this slide a known distance, we were able to compare a baseline camera displacement to the displacement indicated scaled SLAM odometery. The slide allowed movement

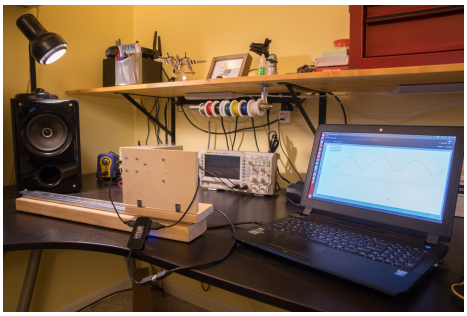
to be constrained to a particular direction to make the displacement easier to visualize, and to allow for controlled testing of various movement angles.



(a) Test System Mounted on Slide



(b) Large Scale Test Environment

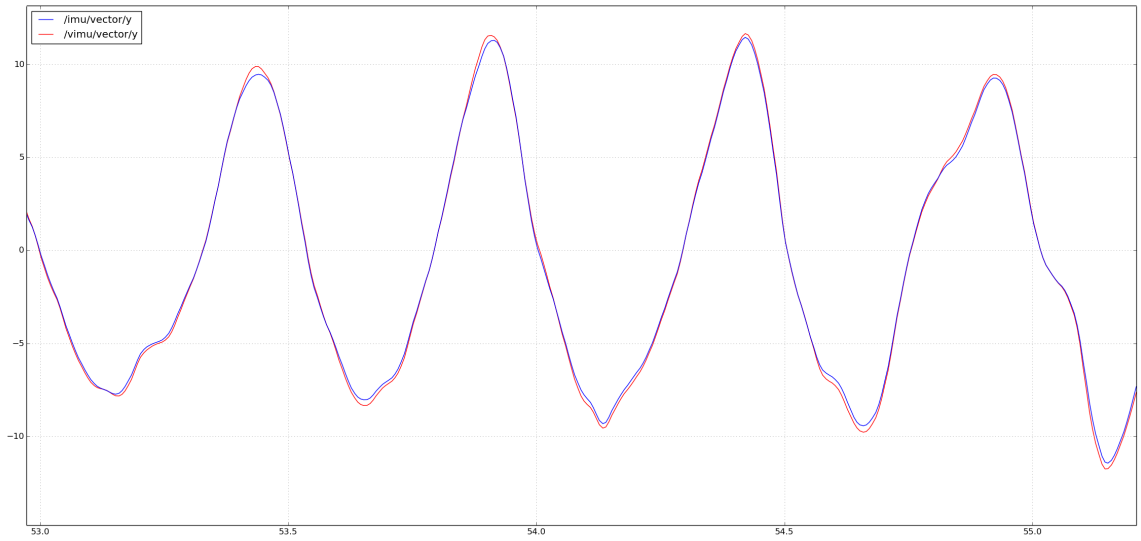


(c) Small Scale Test Environment

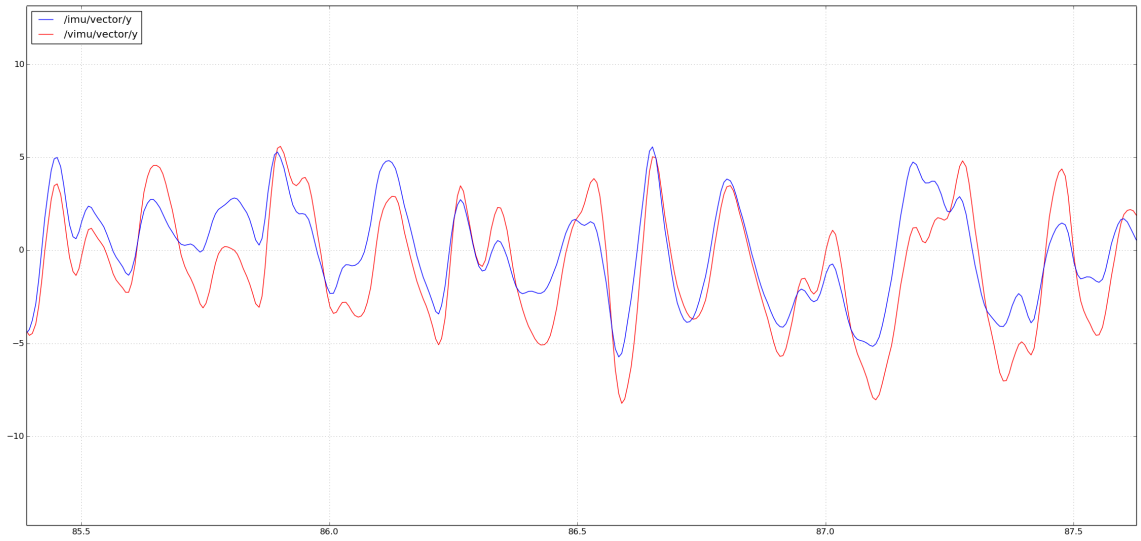
Fig. 4.1. Test Configurations

4.2 VIMU Tests

A simple test was performed on the VIMU to determine whether it had a significant impact on the final signal. The camera and IMU board was detached from the slide, and the board was hand shaken, first without any rotation and then with a twisting motion. The y axes from the VIMU and IMU where graphed for both motion types.



(a) Linear Movement



(b) Rotional Movement

Fig. 4.2. In this test, the IMU was mounted about 2.5 centimeters from the Camera. The VIMU is the inferred acceleration at the point of the camera based on measurements taken at the IMU.

As expected, generally linear motions show very little difference between the IMU and the VIMU. However, twisting motions show significant divergence between the VIMU and IMU. These findings were impressive, given that in this case, displacement

between the IMU and VIMU was only 2.5 centimeters. This is a strong argument against ignoring accelerometer displacement in inertial systems.

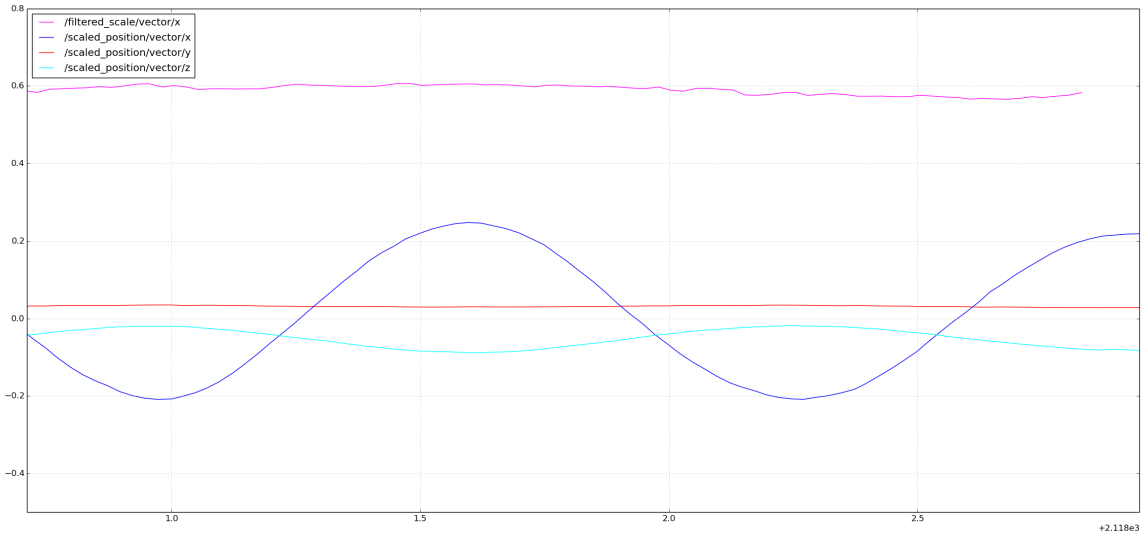
4.3 Displacement Tests

4.3.1 Horizontal Displacement

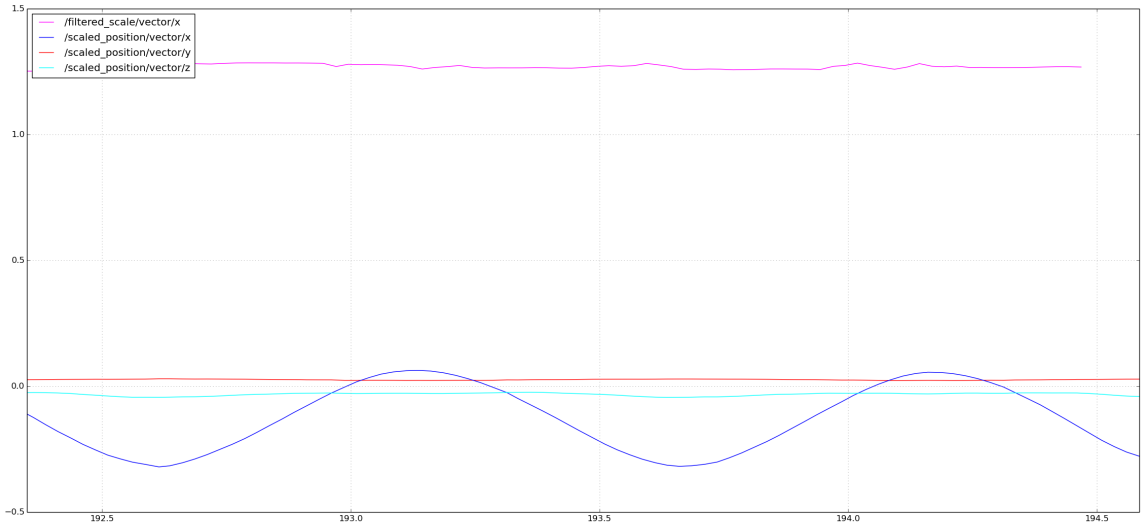
Horizontal movement is a good comprehensive scaling test, because the gravity vector is almost at ninety degrees to the direction of acceleration. This puts $\vec{a}_a$ right at the edge of the zone of allowable values as in Figure 2.4. This means that system noise causes frequently causes the alternate calculation path to be taken. In practice, we observed this to happen about half the time in the horizontal position.

For this test, we marked both ends of a 0.4 meter length on the slide and rapidly slid the camera back and forth between both marks, in order to produce sufficient acceleration to get a stable scale estimate. We tested the system on two differently sized environments in order to be sure that the results were not anomalous. For a large scale environment, we sat the camera slide on the floor in the middle of the room, where the average depth of the scene was approximately 1.5 meters. For a small scale environment, we set the camera slide on a desk and pointed it at the wall for an average scene depth of about 0.5 meters.

We can see that the scale estimate is reasonably stable over a couple of cycles. In both cases it correctly adjusted the internal SLAM odometery to indicate a displacement of approximately 0.4 meters. Both cases slightly overestimated the distance, but not to an extreme degree. Lateral acceleration did have to be fairly large and smooth for the scale to stabilize, but when it did, it always stabilized to the same place.



(a) Small Scale Environment

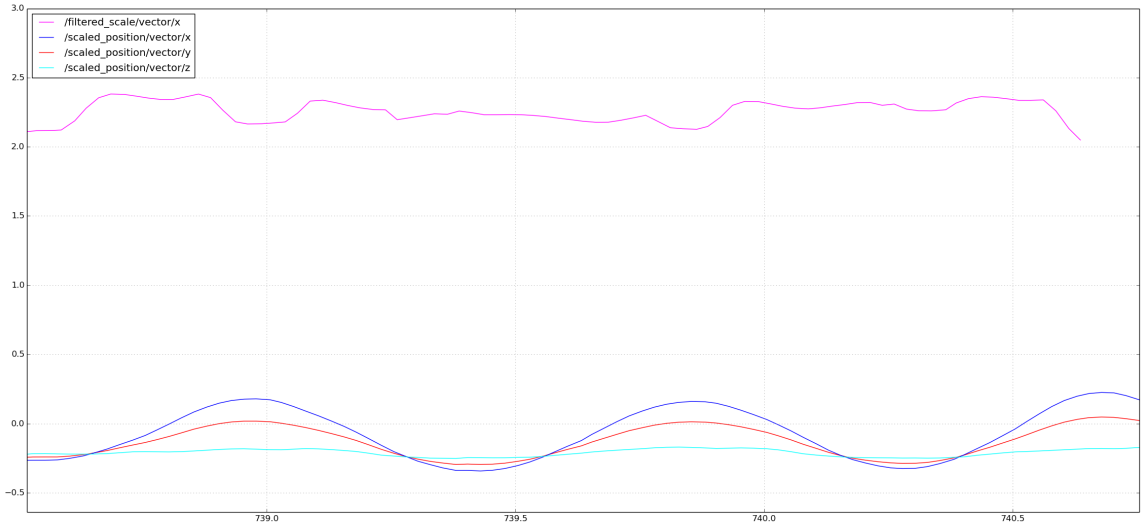


(b) Large Scale Environment

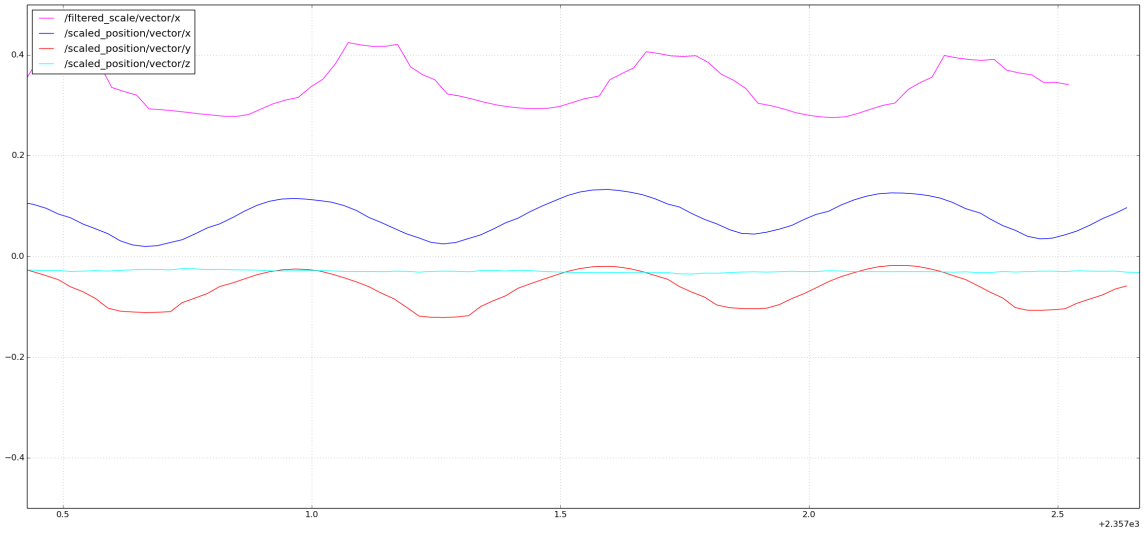
Fig. 4.3. Scaled horizontal position test. Shows x, y, and z displacement plus scale.

4.3.2 Angled Displacement

To test the system with another type of movement, we elevated one end of the slide and repeated the tests for large and small environments. With the slide elevated,



(a) Small Scale Environment



(b) Large Scale Environment

Fig. 4.4. Scaled slanted position test. Shows x, y, and z displacement plus scale.

gravity was no longer at a right angle to movement, so the scale was mostly calculated with the standard method.

The results were not as good with the slanted tests. The compound slide impacted part way through the displacement causing some high frequency distortion in the

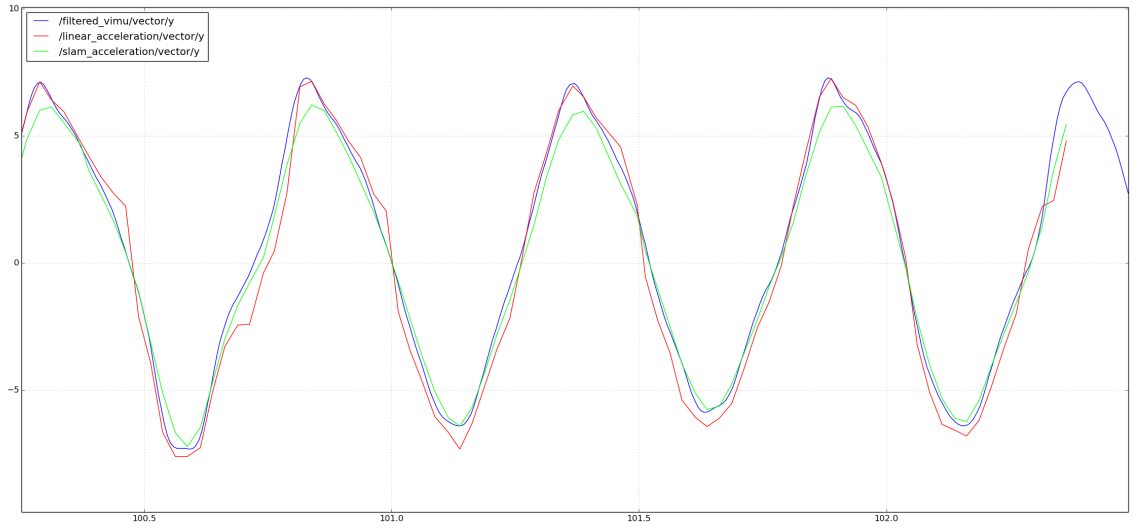
VIMU signal which wasn't completely attenuated by filtering. This was especially apparent in the large scale test, however, it seemed to be a bit more accurate, as the small scale test tended to overestimate scale. It is also likely that the instability of the slanted configuration was more of a challenge for the SLAM odometry.

4.4 Linear Acceleration

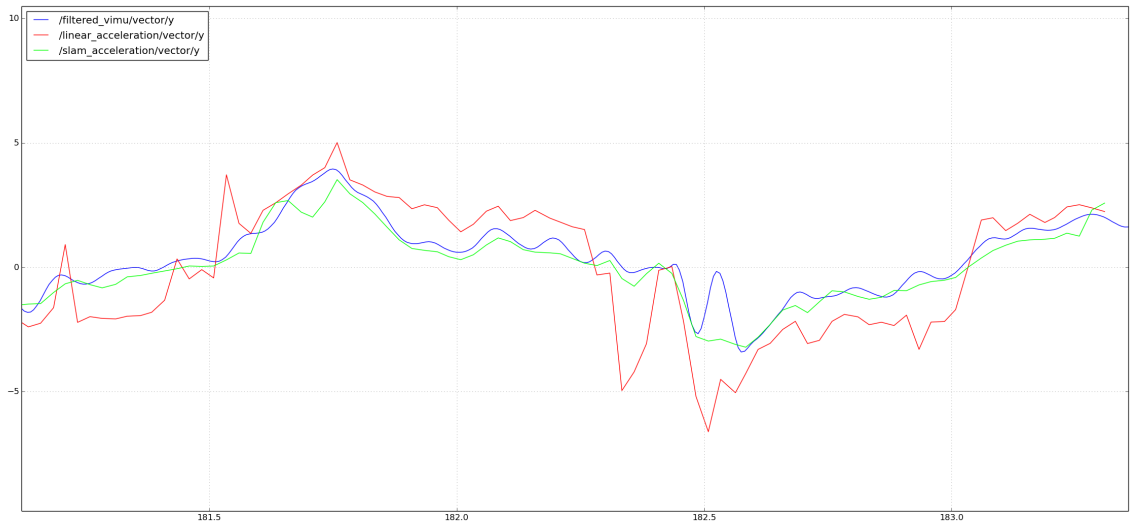
To get better insight into the factors effecting the accuracy of the scale, we took a look at how at the scaled linear acceleration estimate performed. Placing the system in the horizontal position, the horizontal axis of the linear acceleration should be the same as the horizontal axis measurement of the accelerometer, creating a baseline comparison. Note this in this case the horizontal axis is the y axis, while in the positional tests, the horizontal axis is the x axis. This is due to the fact that the SLAM odometry uses a different coordinate frame than the SLAM scaler.

This test strongly reflects the performance of the gravity vector estimate, as the gravity vector is the accelerometer measurement minus the linear acceleration measurement.

We plotted graphs of the scaled linear acceleration, the VIMU, and also included the unscaled SLAM acceleration for reference. In this case the scale factor happens to be very close to one, though that is just coincidence. For this implementation, it appears that the distribution of frequency components in the system motion are very important. The SLAM odometer didn't track higher frequency motion particularly accurately, and the VIMU probably needs to filter higher frequency noise more aggressively. In this case, an IIR filter might be necessary to achieve sufficient filter performance without excessive signal buffering.



(a) Good Performance



(b) Poor Performance

Fig. 4.5. Linear acceleration test. Performance is heavily dependent on the quality of the SLAM odometry. Under good conditions, the estimate is fairly accurate, but as the noise increases, the linear acceleration estimate rapidly deteriorates.

5. CONCLUSION

The results of these tests indicate that the proposed scaling and gravity detection method is viable and promising. However, they also underscore a need for fairly high accuracy from all components of the system to be useful. The single largest limitation of performance is likely the SLAM odometery data rate. One of the most obvious applications for this method is hand held devices, but the 40hz framerate of the camera used in the implementation cannot quite capture enough high frequency detail in hand held motion. It seems likely that a framerate increase to even 60hz would make a big difference.

During testing we noticed that seemly small sources of error can make a surprisingly large difference in the accuracy of the output. For example, at one point it was discovered that a driver issue caused the timestamps on the IMU and the camera to be offset by about 12 milliseconds, which is only about half the period of the camera framerate. However, this offset was enough to completely destabilize the linear acceleration and scale estimates. During testing, the precise amount of delay drifted around by a few milliseconds and would cause issues unless re-calibrated. A number of other issues in the test implementation could easily corrected in a slightly higher budget implementation. The mounting of the camera in relationship to the IMU is very imprecise, which decreases the accuracy of the VIMU. The IMU used had no calibration software, and as a result only had a very crude manual calibration. Correcting these issues, in addition to increasing the camera frame rate, could all improve performance.

It is important to remember what this method needs to accomplish. Right now, it constantly generates a new estimate for the location of gravity with each new frame. The accuracy of the gravity estimate in turn determines the accuracy of the scale. This isn't a realistic approach for a real world system. This is more of a initialization,

because it does not require a-priori information, but as a-priori information becomes available, it should be used. The obvious extension of this approach would be to use the gyroscope and SLAM odometery to fuse multiple gravity estimates over time with a Kalman filter. This gravity estimate could then, in turn, be combined with estimates from other methods. A system implemented in this way could potentially offer a significant advantage in accuracy and robustness compared to current scaling schemes.

REFERENCES

REFERENCES

[1] K. Konolige, M. Agrawal, R. C. Bolles, C. Cowan, M. Fischler, and B. Gerkey, *Outdoor Mapping and Navigation Using Stereo Vision*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 179–190.

[2] F.-e. Ababsa and M. Mallem, "Robust camera pose estimation using 2d fiducials tracking for real-time augmented reality systems," in *Proceedings of the 2004 ACM SIGGRAPH International Conference on Virtual Reality Continuum and Its Applications in Industry*, ser. VRCAI '04. New York, NY, USA: ACM, 2004, pp. 431–435.

[3] Y.-W. Chow, R. Pose, M. Regan, and J. Phillips, "Human visual perception of region warping distortions," in *Proceedings of the 29th Australasian Computer Science Conference - Volume 48*, ser. ACSC '06. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2006, pp. 217–226.

[4] R. Maier, J. Stckler, and D. Cremers, "Super-resolution keyframe fusion for 3d modeling with high-quality textures," in *2015 International Conference on 3D Vision*, October 2015, pp. 536–544.

[5] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *European Conference on Computer Vision (ECCV)*, September 2014.

[6] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.

[7] A. J. Glover, W. P. Maddern, M. Warren, S. Reid, M. Milford, and G. Wyeth, "Openfabmap: An open source toolbox for appearance-based loop closure detection," *2012 IEEE International Conference on Robotics and Automation*, pp. 4730–4735, 2012.

[8] H. Kato and M. Billinghurst, "Marker tracking and hmd calibration for a video-based augmented reality conferencing system," in *Proceedings of the 2Nd IEEE and ACM International Workshop on Augmented Reality*, ser. IWAR '99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 85–.

[9] S. B. Knorr and D. Kurz, "Leveraging the user's face for absolute scale estimation in handheld monocular slam," in *ISMAR*, 2016.

[10] J. Kruttiventi, J. Wu, and J. I. Frankel, "Obtaining time derivative of low-frequency signals with improved signal-to-noise ratio," *IEEE Transactions on Instrumentation and Measurement*, vol. 59, no. 3, pp. 596–603, March 2010.

[11] *IMU Errors and Their Effects*, NovAtel, February 2014, rev. A.

[12] M. Pedley, *Tilt Sensing Using a Three-Axis Accelerometer*, Freescale Semiconductor, March 2013, rev. 6.

[13] B. Fan, Q. Li, C. Wang, and T. Liu, "An adaptive orientation estimation method for magnetic and inertial sensors in the presence of magnetic disturbances," *Sensors*, vol. 17, no. 5, p. 1161, 2017.

[14] T. Kane and D. Levinson, *DynamicsTheory and Applications*. Ithaca, USA: Cornell University Library, 2005.

[15] J. B. Bancroft and G. Lachapelle, "Data fusion algorithms for multiple inertial measurement units," *Sensors*, vol. 11, no. 7, pp. 6771–6798, 2011.

[16] A. Savitzky and M. J. Golay, "Smoothing and differentiation of data by simplified least squares procedures." *Analytical chemistry*, vol. 36, no. 8, pp. 1627–1639, 1964.

[17] *NUCLEO-XXXXRX*, ST Microelectronics, November 2016, rev. 8.

[18] *Motion MEMS and environmental sensor expansion board for STM32 Nucleo*, ST Microelectronics, May 2015, rev. 4.

[19] *iNEMO inertial module: 3D accelerometer and 3D gyroscope*, ST Microelectronics, November 2014, rev. 3.

[20] *UI-1221LE-M-G*, IDS, September 2017.

[21] R. W. Schafer, "On the frequency-domain properties of savitzky-golay filters," HP Laboratories, Tech. Rep., September 2010.