

Localization for FRC

Jinan (Dorothy) Hu, Peter Mitrano, Kacper Puczydlowski, Nicolette Vere*

April 14, 2018

Abstract

1 Introduction

1.1 Motivation

Knowing the position and orientation of a mobile robot is critical to many tasks, and while there are tried-and-true solutions to many instances of the general localization problem, there remain some environments which are unsolved. In this work, we focus specifically on FRC as a test environment. FRC is a challenging environment because the robots make rapid and aggressive maneuvers by human drivers for part of the time, and at other times are under complete autonomy. Another challenge is that FRC fields are cluttered with other robots and game pieces such as soccer balls, pool noddles, or inflatable shapes, and these objects change from year to year. A successful localization system for FRC must support up to six robots, and must be robust to occlusion from the playing field elements, unpredictable lighting, and frequent impacts. Our research suggests that there are at least five appropriate methods for localization: cameras and tags, radio and ultrasonic beacons, optical flow, dead reckoning with encoders, and dead reckoning with an IMU. All of these methods have seen success in robot localization. Generally, FRC robots do not localize to the field. There are a few teams which do attempt this [2], but they lack rigorous analysis and do not compare different methods for localization—their concern is to build something that simply works well enough for their robot and their team. Therefore, we saw the need for a more comprehensive study of localization techniques applicable to GPS-denied environments with fast moving robots and clutter.

1.2 Problem Statement

We are interested in a system for determining the pose consisting of x and y position and orientation θ of the robot. At the highest level, the goal is to allow robots to query their absolute pose on the field. This information is a prerequisite for interesting behaviors such as path following, picking up game pieces, and navigating to feeder stations. An important criteria for our system is that it work well not only on the FRC field, but in teams practice spaces. This means we cannot rely on an *a priori* map of the geometry of the environment, or any other knowledge of the lighting, sound, and surface conditions.

*equal contribution from all authors

1.3 Key Contributions

We contribute a survey of localization techniques applicable to FRC-like environments, a set of metrics which define a successful localization system for FRC, a suite of 24 experiments (see *Experimental Results*) spanning many different different sensing methods, a dataset of robot sensory readings and associated ground-truth positions, and a sample implementation of a full localization system based on all this knowledge. We also evaluate the MarkerMapper tool for building large maps of many ArUco fiducial markers, and provide an array of tools for collecting and analyzing encoder, IMU, and camera data.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Problem Statement	1
1.3	Key Contributions	2
2	Survey of Localization Techniques	3
2.1	LIDAR Mapping	4
2.2	Ultrasonic Mapping	5
2.3	IMUs and Encoders	5
2.4	Beacon systems and Wireless Networks	6
2.5	Cameras with Visual Tags	7
2.5.1	ArUco and MarkerMapper	7
2.6	Optical Flow	8
2.7	Filtering and Calibration	10
3	Trade-Off Analysis Of Different Techniques	10
3.1	Proposed Localization Techniques	14
4	Defining Successful Localization in FRC	14
5	Experimental Results	15
5.1	Double Integration of Accelerometer is Inaccurate	15
5.2	IMU Calibration	16
5.3	Accuracy of Gyro Integration versus On-Chip Yaw Calculation	19
5.4	Characterising Drift and Bias in the Accelerometer	20
5.4.1	Measuring the drift and bias in the accelerometer	21
5.4.2	Zero Velocity Updates	24
5.4.3	Drift Compensation	25
5.5	Comparing Our IMU Localization to the NavX API	27
5.6	Measuring Beacon Delays	28
5.7	Measuring Frequency Response	30
5.8	A Theoretical Procedure for Building a Map of Beacons	31
5.9	OpenCV Optical Flow Sample Code	33
5.10	Benchmarking OpenCV Processing Times	33
5.11	Collecting Ground-Truth with VICON Motion Capture	34
5.12	Detecting Simulated Chirps in MATLAB	35

5.12.1	The Doppler Effect on Ultrasonic	36
5.12.2	Effect of Chirp Bandwidth	38
5.13	Ultrasonic Beam Angle	38
5.14	Characteristics of Piezo Transducers	38
5.15	Co-Processors for Image Processing	38
5.16	Evaluating The Placement of ArUco Tags	38
5.17	Statistics of CSCore Image Timestamps	40
5.18	Effect of Frame Rate and Resolution on ArUco Tag Detection	42
5.19	Rate of position estimates from ArUco Tags	43
5.20	Benchmarking ArUco with VICON Motion Capture	44
5.21	Benchmarking MarkerMapper with VICON Motion Capture	44
5.22	Our Experiences with Building MarkerMaps	45
5.23	Erroneous detections with ArUco	46
5.24	Latency over the Robot Network	47
6	A Dataset for Robot Localization	48
6.1	Provided Tools	49
7	Sample Implementation	49
7.1	Sensing Techniques Used	49
7.2	Robot Hardware	49
7.3	Kalman Filtering	50
7.3.1	Accelerometer Pre-Processing	50
7.3.2	Encoder Pre-Processing	50
7.4	Software Design	50
8	Conclusion	51
9	Future Work	51
10	Acknowledgements	51
11	Appendices	55
11.1	Ultrasonic Radio Beacons Bill of Materials	55
11.2	Survey Responses	55
11.3	Radio Time of Flight	56
11.4	ArUco Detection Times	56
11.5	Code & Dataset	57

2 Survey of Localization Techniques

In this section we provide an overview of the most common and applicable localization techniques. Overall, the problem of localizing a mobile robot can be viewed as accurately measuring the absolute distance to known landmarks, or by measuring the changes in position over time. All localization methods lie somewhere on a spectrum between these two approaches, and we will henceforth refer to these two ideas as global and local pose estimation. Some of the high level techniques for robot localization are: measuring range at various points around the robot and matching these readings to a map, measuring time

of flight or difference of arrival time to calculate distances to known locations, recognizing landmarks and computing pose relative to those landmarks, and measuring changes in pose and accumulating these changes over time. There are different sensors that can be used for each of these techniques, such as laser range finders, cameras, inertial measurement units (IMU), encoders, radio, infrared light, visible light, ultrasonic and audible sound. Although there are a tremendous number of possible methods for indoor mobile robot localization, there are a few which have received the most attention and shown the most success. These include, but are not limited to:

- LIDAR mapping
- Ultrasonic mapping
- IMU and Encoders fusion
- Infrared or Radio and Ultrasonic beacons
- Wireless network methods based on signal strength
- Cameras with visually identifiable tags
- Optical flow mice and cameras

In our research, we learned how these techniques work and found descriptions and implementations to figure out whether they are appropriate for high-speed, cluttered, multi-robot environments like FRC. These descriptions and implementations are presented in this section with the purpose of demonstrating a thorough literature review and of providing background information to the reader.

2.1 LIDAR Mapping

LIDAR is a sensor that works by measuring the amount of time it takes a laser to return to the LIDAR after hitting a desired object [17]. Since light moves at a constant speed, the LIDAR can calculate the distance between itself and the object that light was hitting. The formula to compute distance is $\frac{d*c}{2}$, where d is the distance to the object, c is the speed of light and the division by two accounts for traveling to the object and back. By repeating this process at different angles the LIDAR can produce a map of its surroundings by finding the distance between it and surrounding objects within its detecting range.

There are three types of information LIDAR can collect depending on the type of LIDAR. The first is the range to the target which is found using a topographical LIDAR. A differential Absorption LIDAR can find the chemical properties of the targets it is measuring. A doppler LIDAR can measure the velocity of a target. For our scenario, we concern ourselves only with topographical LIDAR methods.

Most LIDAR have two main pulse systems for measuring distance. The first system uses a micropulse have lower powered lasers that are usually considered safer [17]. The wavelength for these is typically 1.0-1.5 m [41]. The second system uses high energy lasers and is typically only used for atmospheric measurements [17]. The wavelength of these is typically 0.5-0.55 m [41]. LIDAR localization works by matching landmarks to some known map. Since the distance between it and those landmarks are known, the LIDAR system can be used to determine its own position [35]. Another approach is to match point clouds found on the most recent map produced by the LIDAR to point clouds on the prior map.

This has advantages because it does not rely on there being distinguishing features in the environment. But it also takes more time to compute the map since it has to compare more points than a feature to feature map [22].

2.2 Ultrasonic Mapping

Ultrasonic mapping (often referred to as sonar) was one of the first techniques used for indoor robot localization, and has been explored deeply since the 1980's. The most common approach is to use multiple emitter-receiver transducers placed around the perimeter of the robot, measure the range at each of those points, then localized to a given map of the environment [9]. Alternately, some systems use one sensor and rotate it to achieve the same effect [21, 9]. The algorithms for interpreting the measured distances work by first extracting lines, then matching these lines to an existing map using algorithms such as RANSAC. Reported accuracies of the system in [9] was 1 ft for position, and 10° for angle. In [9] and [21], the reported rate of position updates is 1 Hz. Additionally, some methods will explicitly model the uncertainty of the position estimate, or explicitly model the behavior of ultrasonic sensors to ignore unreliable data. A more recent and sophisticated approach to localizing with sonar can be found in [38], in which 24 sonar sensors in conjunction with encoders were used to perform simultaneous localization and mapping. Their experimental results report drifts of 3.9 m and 21° over the course of 35 m of travel.

2.3 IMUs and Encoders

An inertial measurement unit (IMU) is a sensor reporting the forces acting upon an object, the angular rates of change, and surrounding magnetic field. The device typically comprises an accelerometer, gyroscope, and magnetometer which sense the above data respectively. These devices function by detecting Coriolis forces, which are inertial forces acting in the direction of motion relative to a rotating reference frame. These forces are proportional to the magnitude of the acceleration. These forces may be detected by a simple strain gauge mechanism or object vibrating at a known frequency (the rate of change of vibration frequency is detected) [3]. The premise behind position sensing using this device involves integrating the data with respect to time to calculate position and orientation. This approach was first used in aeronautics to estimate projectile attitude, orientation, and position [28]. High cost IMUs have been used historically for defense and transportation systems; the quality of the sensor is high and the data is reliable in these applications. An inertial navigation system (INS) often comprises multiple accelerometers, gyroscopes, and magnetometers to estimate orientation and position. Their performance is increased by referencing, or filtering, one sensor to estimate the error from another. Simple double integration of a filtered system using expensive sensors is often sufficient for position tracking applications like ballistics or missile tracking [3].

In cost-sensitive systems, these simple methods are much less accurate because the low-cost electronics have more drift and noise. Because of integration of accelerometer data, the velocity error term grows linearly and position error grows quadratically. This introduces a need for more sophisticated filtering, sensor fusion, and optimization based approaches. Bayesian filters (Kalman Filter, Particle Filter, ...) are one family of filtering algorithms commonly used with IMUs.

If the rate at which the position must be updated is lower than the update rate of the data, many values can be processed and used to calculate an approximation within

a given time window. This technique is known as preintegration. Instead of filtering the data, preintegration combines many data points into a single trajectory estimate. Then, it transforms the data into the navigation frame, allowing for a smoother approximation of system position. This was beneficial in cases where global position data was unavailable for extended periods of time, and it also decreases computational load of the localization thread [25]. The authors of [25] describe an overall CPU time of about 10ms for data processing and real-time execution, although the system update frequency is unknown.

Another method for computing position from IMU data is presented in [42]. The state estimate and sensors measurements, which include imagery in addition to IMU data, are represented as a factor graph, and a novel algorithm is presented to update these estimates to approximately-optimally estimate the true state. The main benefit of this approach is improved computational complexity over methods like Bundle Adjustment, without requiring linear or approximately-linear sensor models like with Kalman or Extended Kalman Filters.

Due to the widespread availability and well understood algorithms for using IMUs to derive position, there exist libraries for IMU based localization already available to FRC team. Frameworks such as Sensor Fusion 2 (SF2) provide students with algorithms that include double integration, latency correction between IMU and camera data, fusion of encoder and IMU data, and keyframe-based state estimation [14]. These algorithms use known system parameters, such as update frequencies of sensors, frame transformations between sensors, and data from landmarks for filtering and position estimation. Additionally, the data is accurately timestamped and easily accessible to the vision processing thread. This way, the user receives an updated pose estimate without lag and has a history of the orientation. However, we suggest that these libraries are not quite robust enough for FRC teams to rely on them for accurate position (see Defining Successful Localization in FRC)

2.4 Beacon systems and Wireless Networks

Beacon systems generally use ultrasound and or radio as a medium and either signal strength, phase shift, or time to measure distance to the beacons. Among radio systems, the system in [1] identified the location of people moving around buildings using signal strength in the 2.4GHz band received at three or more beacons, and they report accuracy of a few meters with an update rate of at most four times per second. The systems described in [8] uses passive RFID tags on the ceiling and an RFID transmitter on the robot, and report an accuracy of 4cm within a 5m². Another RFID system [33] also uses signal strength to RFID, and reports accuracies for various configurations ranging from 1cm to 3m. These RFID systems use readers that cost over \$500.

There are also countless localization systems that use standard wireless networks. A comprehensive survey of these systems can be found in [24]. Systems that use signal strength in standard wireless LAN networks have achieved up to 10cm accuracy and hundreds of updates per second. Another radio beacon solution is to substitute single-frequency radio with Ultra-wideband radio. These systems can achieve centimeter level accuracy, but they use obscure or custom made transmitters and receivers that cost in the hundreds of dollars [47] [31].

Among ultrasonic beacon systems, [19] uses the raw arrival times of ultrasonic pulses over time plus odometry together in a Kalman filter. Many beacon systems use the speed difference between sound and electromagnetic waves to measure system. Systems like [36], [44], and [18] send radio pulses followed by ultrasonic pulses. This is known as the “Cricket” style of beacons. Nodes in the network us the difference in arrival time of these two signals

to measure distance. Alternately, some systems use infrared pulses in place of radio [12] [46]. These systems are inexpensive, and report accuracy of between 2 cm and 14 cm.

In the remainder of this paper, we will always be referring to the “Cricket” beacon localization method. This method has been shown to be accurate and affordable, and as we will discuss in the Trade-Off Analysis Of Different Techniques section, it nicely complements our other proposed methods of localization.

2.5 Cameras with Visual Tags

Most methods for indoor localization assume some amount of either natural landmarks or artificial landmarks in the environments as references to absolute positions. In either case, the general approach is to calculate the pose of the robot with respect to one or more landmarks, then use the known position of the landmarks to calculate the pose of the robot in a global frame. Another similar approach is using 3D models and 2D to 3D matching techniques. The system described in [34] had accurately localized the camera’s position using this 2D to 3D mapping technique. The most common method for localization is artificial landmarks. Common artificial landmarks include 1D binary barcode, 2D binary barcode and 2D colored barcode. The system in [23] used cameras and ID tags on the ceiling, which were 2 m away from the floor. A web camera facing the ceiling was mounted on a moving robot with a speed of 20 cm s^{-1} . The result of the experiment in [23] showed that this method was accurate even though there was an unevenness between the ceiling and the floor. Another system [13] also used camera and tags. However, instead of sticking ID tags on the ceiling, it put invisible tags on the floor by every 3.75 cm. The camera it used was surrounded by a UV light, which allowed the camera to capture those invisible tags. This system performed really well in homelike environments, and the authors report only a few centimeters of error. Another barcode based localization system for robots with very limited memory and computational resources (8 KB memory, 16 MIPs) [7] used 1d barcodes as references. Using a camera with 80 vertical pixels and 640 horizontal pixels, the system achieved localization within 7.8 cm of error on average. Ultimately, cameras with artificial visual markers have been shown to be accurate enough for our application (see section 4) but are highly dependant on the various assumptions about tag placement, camera quality, processor capabilities, and the required frequency of position estimates.

2.5.1 ArUco and MarkerMapper

ArUco is one implementation of artificial landmark based localization that has been used extensively in robotics research. The ArUco library (<https://sourceforge.net/projects/aruco/>) provides a function for estimating the pose of an object by minimizing the squared sums of the distances between the projected points and the measured points (reprojection error). The side length of each tag is known and input into the program. The measured points (two corners, minimally) are used to obtain a point estimate in 3D space. Multiple point estimates from each corner are used to calculate the pose of the ArUco tag’s centroid. The projected points are parameterized by the camera matrix, which uses the pinhole camera model. The reprojection error corrects the pose estimate based on the calibrated values. An example of a correctly detected ArUco tag can be seen in Figure 2.

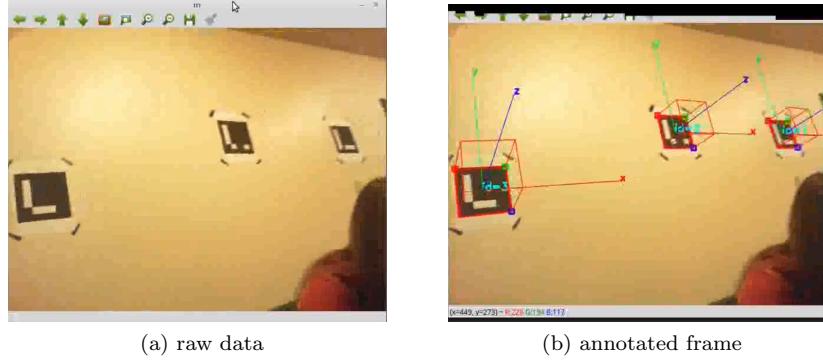


Figure 2: Visualization of pose estimate

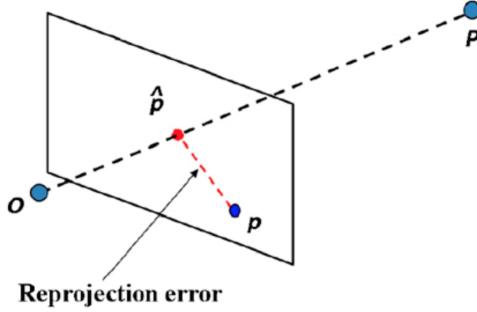


Figure 1: Calculating reprojection error [32].

MarkerMapper (<https://sourceforge.net/projects/markermapper/>) builds on top of the detection of individual markers. Markers are first scattered around the environment, then MarkerMapper can be used to build a map of their poses in space. The ID of the origin is input into the program. By estimating the pose of each tag in a camera frame, a map of transforms between tags was developed. Then, the position of the robot's camera, and by extension the robot itself, can be found with respect to detected markers. When the map is complete, the user is able to query the pose from the origin using data from any tag in the workspace. This approach has the advantage of not requiring tags to be placed carefully at known locations, which is a difficult problem in cluttered environments.

2.6 Optical Flow

Optical flow is the ability to track changes between cameras frames and measure the differences between them to track position. In other words, optical flow is a collection of techniques for finding the movement of objects between images or video frames. More precisely, optical flow looks at the movement of pixels among images. There are many assumptions about the image that has to be made in order to apply optical flow. The first is that the lighting in the image stays consistent throughout the sequence of images. Images with inconsistent lighting or transparent objects would violate this assumption. Limiting

the amount of inconsistencies in each sequence of images leads to more accurate optical flow.

There are many methods of calculating optical flow that deal with different constraints. This first is the Horn and Schunk method which calculates optical flow looking at all pixels in an image. Methods which consider all the pixels are called global methods. Along with the lighting constraint it also adds that the image should be as smooth as possible and have few variations in its coloration. The closer the amount of variations is to zero the more accurate the optical calculation will be[29].

The optical flow vector for each pixel is calculated using the equation below. I_x and I_y are the spatial gradient of the current pixel. Spatial gradient refers to the path the pixel is moving along. I_t is the temporal gradient of the current pixel. Temporal gradient is how similar the motion of the pixel is to its neighbors [37]. α is a weighting term. \bar{u} and \bar{v} are the components of the average optical flow vector of neighboring pixels. The equation is shown below 1 [29]. n represents which iteration the optical flow calculation is on. Each current pixels' optical flow is calculated based on the optical flow of the pixels at the previous iteration. Optical flow calculation will iterate from pixel to pixel until it has calculated optical flow for each pixel.

$$\begin{aligned} u^{n+1} &= \bar{u}^n - \frac{I_x[I_x\bar{u}^n + I_y\bar{v}^n + I_t]}{\alpha^2 + I_x^2 + I_y^2} \\ v^{n+1} &= \bar{v}^n - \frac{I_y[I_y\bar{u}^n + I_x\bar{v}^n + I_t]}{\alpha^2 + I_x^2 + I_y^2} \end{aligned} \quad (1)$$

Optical flow can also be done locally using the Lucas Kanade method [37]. This method is based on the assumption that the optical flow vector of pixels are similar to their surrounding pixels. This method finds optical flow vectors that are consistent with its neighboring pixels' temporal gradients and spatial gradients. Each neighbor is then given a weight based off of how close it is to the pixel. The farther away a pixel is, the lower a weight it is assigned. This is because spatial and temporal gradients are based on how far away a pixel is so the error will be larger. Having a lower weight will reduce the error. The formula for the optical flow vector is a least squares equation shown below in equation 2 [29].

$$E_v = \sum_{p \in \Omega} W^2(p)[\nabla I(p) \cdot v + I_t(p)] \quad (2)$$

$\nabla I(p)$ and $I_t(p)$ are the spatial gradient and the temporal gradient for each of the neighboring pixels p . v is the optical flow vector for pixel located at (x, y) on the image. $W(p)$ is the weight assigned for each pixel. Local methods tend to work better since they do not allow information about vectors to spread to unrelated regions of the image. This issue of information spreading to unrelated areas of the image is especially problematic in global methods when the assumptions about consistent smoothness and illumination are not fully met. There are a variety of other optical flow methods that focus on different ways of comparing pixels within images but local and global are the most popular methods [29].

Optical flow has been used for multi-sensor localization in indoor, feature-rich environments [11]. This method is also sometimes called visual odometry. In this work, the authors use a PX4FLOW optical flow sensor to capture 64x64 pixel images at 100 FPS, and an ultrasonic range sensor to measure distance from the ground. The data from the camera was used to obtain a velocity information using optical flow and a position estimate using landmark detection on the images. These were fused with attitude data from an onboard IMU.

In this research, miniature quad-copters flying over a textured carpet are used to evaluate the localization algorithm. The patterns on the 20x20m carpet, comprising dots of random size and a 1 square grid, are used as features for the optical flow and camera-based position estimates. The authors report average error of 0.025 m in a test of stationary hovering.

2.7 Filtering and Calibration

Given the number of sources of position information, it is natural that there will also be a number of ways to take advantage of using multiple techniques together. Different sensors can have better or worse performance in different scenarios, and a choice of fusion algorithm will yield more accurate position information by leveraging this. Calibration can also be used to compensate for errors between sensors. For instance, if you have encoders to determine that you are not moving, you can take your current IMU readings as a bias and using them to reduce the error build up during integration. The most popular class of filtering algorithms used for localization is called Bayesian filters. These filters describe the world and sensors with probability models, and they estimate both the state of the robot and the confidence (covariance) of that state estimate. Bayesian filtering algorithms include Kalman filters, information filters, and particle filters. Kalman filters and information filters have the advantage in computational efficiency, where as particle filters can more be more accurate if the true belief distribution is non-gaussian or if the true dynamics are nonlinear [40]. In our work, it is natural to consider the state as the position, velocity, and acceleration of the robot. It is common to assume that this state representation satisfies the Markov condition needed by Bayesian filters. Intuitively, the Markov condition says that knowing our current state and control input is sufficient to make a prediction of the next state, and we do not need the full history of states and control inputs. To implement these filters, we required a model for how our state changes given our current state and motor inputs. For each measurement source, we define how the sensor values are derived from the state. It is easy to come up with very rough approximations for these equations, but difficult to construct accurate ones. On the other hand, these filters have very strong guarantees and their efficacy has been demonstrated in numerous systems [5][8][26][28][33].

Many of the localization techniques discussed involve some form of calibration. Primarily, the IMU requires calibration for misaligned axis, scaling factors, and biases. There are many procedures for calculating these calibration parameters by taking advantage of static intervals and assumptions about the force of gravity [25][20][39]. Visual tag detection algorithms, such as ArUco, also include a camera calibration process to account for the focal length, field of view, and distortion characteristics of the camera [15]. Knowing these parameters allows one to undo distortion to the image, which is essential for detection of most AR tags.

3 Trade-Off Analysis Of Different Techniques

Each of the techniques presented thus far have strengths and weaknesses. In cases where those strengths and weaknesses are orthogonal, combining multiple techniques can improve the overall performance. This is the fundamental principle behind sensor fusion. For example, in [18] the authors use a compass to make up for the inability of beacons to measure orientation of the robot. In order to tackle all of the diverse challenges of localization in the FRC environment, we believe it is necessary to combine techniques. In this section we will explain which techniques we are promising and which we have ruled out. We will justify

why none of the techniques discussed are sufficient on their own, and explain which the techniques we have chosen work well together.

As stated in section 2, techniques for localization include LIDAR mapping, ultrasonic mapping, IMU and encoders, infrared or radio and ultrasonic beacons, wireless network methods, cameras with tags, and optical flow. Each of these techniques has been used successfully in their respective applications, but not all of them are appropriate for this project.

LIDAR has been shown to be one of the highest performing localization methods in terms of accuracy, precision and update rate. The two reasons why we are not pursuing it further are because it is too expensive and because it requires a map. LIDARs capable of ranging across an entire FRC field are over \$400, which is the cost limit for any single part on an FRC robot. Additionally, LIDAR techniques also require either mapping on the fly, or an existing map. Mapping on the fly presents its own challenges, and usually suffers from very bad localization for some initial period of time while the map is built. Therefore, a map would have to be provided for the environment. Existing maps would work very well on the competition FRC fields, but would not apply in the practice spaces teams use because their practice spaces change frequently, and building and maintaining useful maps in those spaces would be a burden.

Ultrasonic mapping has this same issue. Both LIDAR and ultrasonic mapping would work best if teams to place walls up to create a “pen” for the robot of known geometry to use as a map, and for this reason we believe LIDAR and ultrasonic mapping are unfit. Another major issue with ultrasonic mapping is the interference between robots. If multiple robots range ultrasonic near one another, there could be cross talk and interference between the signals. This is reason enough to rule out any use of reflecting ultrasonic. Note however that ultrasonic beacons do not have this weakness, since the pulses being emitted are being timed based on line-of-sight travel with so any reflections can and should be ignored.

IMUs within the budget of FRC teams suffer from accumulated drift, and as such they cannot be used in isolation (see 5.1). On the other hand, many FRC students have experience with them, so it would be wise to support basic features such as heading detection and filtering using IMUs. IMUs also compliment other localization techniques very well. For example, cameras suffer from the jitter of the robot moving, and encoders fail when the wheels slip. IMUs on the other hand are excellent at detecting jitter and slippage. In this way, an IMU is a good complement to cameras and encoders.

Radio and ultrasonic beacons are very attractive because of their low cost and ability to automatically locate each other. The cost of each beacon are projected to cost about \$30 (see 12). Furthermore, beacons have more flexibility in their placement than tags because they are much smaller and do not need to be on flat surfaces, or in specific orientations. Finally, because each beacon can operate as a transmitter or a receiver, beacons can automatically locate each other, which means students will not have to measure their positions or worry about them being accidentally bumped. A procedure for building a map of beacons is described in section 5.8. Beacons also make up for some flaws in the other techniques. Beacons provide absolute global position but updates slowly, which nicely complements IMU and encoder methods which are fast but only measure changes in position. Additionally, beacons are more resistant to jitter than cameras. Finally, by placing the beacons and cameras in different locations we can minimize the effect of occlusion.

Wireless network systems are among the most popular for indoor localization. However, they also require knowledge and control over the 2.5 GHz spectrum in the area where they are used. At FRC events, there can be dozens of wireless networks running, as well as the

wireless networks used on the field for communication between robots. For this reason, we feel that techniques using wireless frequency have too many unknown variables. It's possible that there are methods other than signal-strength 2.5 GHz based systems which could work well for FRC, but those advanced techniques are neither well established nor within our ability to implement.

Among the vision based localization systems discussed in section 2, there are systems that use natural landmarks (object detection) and those that use artificial landmarks (tags). Tag based systems are preferred because they are inexpensive and easy to implement. Natural landmark detection would likely not perform well in cluttered high-speed environments like FRC because of moving robots and game pieces. Furthermore, implementing real time object recognition is computationally intensive. Among systems using artificial landmarks, not a lot of robot localization systems use 1D barcodes as references. A 1D barcode can only contain up to 25 characters, which limits the length of information. Among 2D barcodes, fiducial tags and QR tags are two of most popular choices in mobile robot localization. The advantages and disadvantages of different types (QR, Data matrix, PDF417, fiducial tag) of 2D barcodes are discussed here. QR codes are designed to be viewed straight on with the camera. Data Matrix codes are very similar to QR codes, and they have high fault tolerance and fast readability. Data Matrix can be recognized with up to 60% of the code unrecognizable. PDF417 is famous for the huge amount of data it can store. Complex information such as photographs, signatures can be inserted into PDF417 easily. Fiducial tags contain less information than QR codes. However, many of them can easily be detected in one shot and the process speed for fiducial tags is faster than of QR codes, and so they have seen widespread adoption in robotics. The system in [43] measured the distance between AprilTags and the camera. A sheet of 16.7 cm AprilTags were tested from 0.5 m to 7 m away. The calculated distance was within 0.1 m of the real distance from 0.5 m to 6.5 m. However, orientation errors were pretty high (1.5° off) when the off-axis angle was small, but were within 1 degree from 20° to 75° of off-axis angle. The detected rates for tags were 100% from 0 to 17 m away. This system showed that the combination of camera and fiducial tags can potentially localize robots accurately and precisely. In [4], the authors developed an algorithm to enhance the quality of QR codes captured in order to improve the recognition rate. Its algorithm successfully recognized 96% of QR codes under a variety of qualities captured by a mobile phone camera. The average time for decoding a QR code is 593 ms. Another deblurring method in [45] can be applied to enhance the quality of motion-blurred ArUco code.

Another benefit of cameras with tags is that they provide global position information without much setup or infrastructure. However, camera based systems suffer from occlusion and jitter. These disadvantages can be mitigated with our other localization techniques. In summary, tag based camera systems have been shown to be accurate enough for use in FRC, and it complements other localization methods well. Marker Mapper is localization technique for indoor robots published by the developers of the ArUco tag detection and pose estimation algorithm. Motion capture data suggests that it is comparable to sophisticated localization algorithms such as ORB-SLAM and LSD-SLAM[27].

<i>Sequence</i>	Ours	LSD-SLAM	ORB-SLAM2
SLAM-Seq 1	0.0447	0.440	0.231
SLAM-Seq 2	0.0433	0.117	0.913
SLAM-Seq 3	0.0694	0.652	0.314

Figure 3: Marker Mapper absolute trajectory error (meters)

The algorithm must first construct a map using off-line data. Once the transforms between tags are known, the map is used to report position from a known tag. The transforms between tags are corrected using redundant information in frames. The error along each basis cycle is computed, then an optimization algorithm is used to compute the corrected pose estimation. The mapping phase is an order of magnitude faster than Structure from Motion (SFM) and Multiple View Geometry (MVG) localization techniques. Although the paper mentions no on-line tests, is it reasonable to believe that pose estimation can be accomplished at minimally a 1Hz rate.

Optical flow offers accurate angle measurements and fast updates that are relative to our current position. Like all camera based solutions, the vibration of the robot will likely makes this technique difficult. However, cameras are the most widely used sensor according to our survey of FRC students and alumni, which is another benefit of optical flow and tag based solutions. Optical flow can be applied either to cameras facing the environment or pointed down at the floor.

The latter is the method used by computer mice, which have optical flow chips designed for high speed motion. Optical flow chips are made for optical flow detection with a specific lenses and microprocessor to get position [10]. These types of chips are built into computer mice with lenses that work only when the mouse is against a flat surface at a specific height from the table. This would be a problem in FRC since the field is not perfectly flat and there are sometimes obstacles that the robots need to drive over. There are also different drive trains which can shift center of balance between sets of wheels which would also cause the mouse to be off the ground. One of the benefits of using a mouse would the fast update rate. Optical flow mice update at 2,000 to 6,469 frames per second according to the ADNS-3080 optical flow sensors specifications [37]. They process frames quickly and most teams have mice of some sort they could use. However, a drawback of optical flow mice is their inability to detect rotation. Any rotational component in the optical flow is explicitly removed since computer users want only the translation of the mouse in order to navigate a computer screen. Lighting is also important to for the camera to be able to clearly pick up images so having a source of light illuminating around the optical flow mouse would also be necessary for teams in order to get the best results [10].

The other option for optical flow is to use a camera which faces the environment. This method is also sometimes called visual odometry. OpenCV provides libraries and sample programs for running dense optical flow and sparse optical flow in these configurations. Dense optical flow takes longer since it is using all of the points on a frame but can be more accurate [16]. In general, optical flow is not sufficient for localization on its own because it does not provide position in any global frame. However, environment-facing optical flow nicely complements our other systems because it uses a sensor we already plan to use (a simple webcam), and provides a source of local position updates not based on any assumptions about wheels or robot dynamics.

3.1 Proposed Localization Techniques

Ultimately, we have identified IMUs, encoders, cameras with tags, beacons, and optical flow as promising techniques for localization in FRC. These techniques together provide redundant sources of both local and global pose estimates, and account for many of the challenges associated with localization for FRC. We believe that implementing each of these techniques and combining their results will produce a more robust localization than exploring any one of them in depth.

4 Defining Successful Localization in FRC

Here we present the criteria a system must meet in order to be successful. Broadly, we consider the following factors to be those which are important, since they immediately effect the ability of an FRC team to use localization for interesting tasks.

1. Accuracy

How close our position estimates are to ground truth.

2. Precision

How close repeated position estimates are to each other given the same ground truth.

3. Update Rate

How quickly does our system provide position estimates.

4. Accessibility

How affordable is our system, how difficult is it to make, and how easy is it for teams to use.

A successful localization system for FRC should meet the following criteria:

1. Accuracy of ± 10 cm and $\pm 5^\circ$

2. Precision of ± 5 cm and $\pm 2^\circ$

3. Update Rate at 20 ms/50FPS, with global updates at 100 ms/10FPS

4. Accessibility with cost under \$200 for teams.

To come up with hard numbers for these criteria, we first performed a few simple calculations based on our knowledge of FRC and a survey we conducted. First, we consider what teams would want to use position information for, and decided that the applications requiring the most accuracy are shooting and autonomous pick of game pieces at known locations. Both of these require the position estimates to be close to the true position of the robot. From there, we estimate that most FRC shooting and pickup mechanisms will work within ± 10 cm. Next, we decided the application requiring the most precision would be path following. If position estimates are imprecise and jump around rapidly, smooth path following will be difficult. From our experience with path following, we estimated that ± 5 cm and $\pm 2^\circ$ would be sufficient. For update rate, we considered what the maximum distance a robot could move within a period and used that to decide what our update rate should be. The very fastest FRC robots move 6 m s^{-1} , which at an update rate of every 20 ms is a distance of $0.02 * 6 = 0.12$ m. The rate of 20 ms is a realistic cycle time in FRC,

and we feel 12 cm is sufficient given the speed. For accessibility, we acknowledged that teams cannot spend more than \$400 on any part, and that they usually source parts from websites AndyMark, Cross-the-road Electronics, and National Instruments among other suppliers. We are also conscious that many FRC teams have limited or cluttered spaces for testing their robots, and may be working in a shared space that must be clean and usable after their work sessions.

Using all of these informal estimates as a starting point, we conducted a survey of FRC students, alumni, and mentors. We received 65 responses in total, and used the results of this survey to solidify these design criteria. The full response of this survey are presented in *Survey Responses*. In summary, the median for accuracy was 4 inches in x,y and 5° in yaw. Our survey did not include questions about precision and update rate, because they depend on what position is used for. Instead, we asked if students would try path planning if they had a localization system, which would back up our estimate of precision. Our survey indicated that 90% of students would try to make the robot autonomously follow paths. Therefore, our precision estimated based on path planning as an application is supported by our survey. Update rate was not addressed in the survey because we didn't think FRC students would have informed opinions on this metric. Finally, we asked several questions about the accessibility requirements. A cost of under \$200 was deemed acceptable by 84.6% of responses, and so we have made \$200 the goal for cost. Furthermore, we learned that the amount of space in teams shops varies from a 5 by 5 foot space up to several thousand square feet, but the median shop size is 775 ft², which one can imagine as a 25 by 30 ft space. In terms of access, about 76.5% of teams could leave up tags or beacons, with the others stating that they must clean up everything because they work in a shared space such as a classroom. Lastly, we asked students what sensors they were familiar with. The most familiar sensors were cameras (90%), followed by encoders (84.6%), then IMUs (60%). Therefore, it would be beneficial to incorporate cameras, encoders, and IMUs because teams are already familiar with them. However, in order to not place extra constraints on sourcing parts, we choose to ignore the constraint that the parts we test with meet the FRC-Legal or Off-The-Shelf requirements of FRC.

Ultimately, we formulated design criteria based on our own experience with FRC and with localization, as well as by conducting a survey of the needs, experience, and opinions of FRC participants. These design criteria will help us pick which localization techniques to pursue as well as define a successful localization system for FRC.

5 Experimental Results

One of the key contributions of this MQP is an extensive set of empirical and theoretical results spanning the 5 different sensing technologies we outlined as promising (section 3.1). This section describes each of these experiments and explains how each test impacts the practical implementation of a complete localization system. Future projects working to implement an actual localization system for FRC can use these results to jump-start their development and inform design decisions.

5.1 Double Integration of Accelerometer is Inaccurate

We first demonstrate that double integration of raw accelerometer data is inaccurate. This is unsurprising, but for completeness we demonstrate specifically that double integration is

inaccurate for the NavX IMU under FRC-like driving conditions. This inaccuracy comes from manufacturing errors, and electrical noise and imperfections in the IMU circuitry. Noise is also introduced from the vibrations of the robot chassis as it drives. Figure 4 shows a typical example of naive trapezoidal rule to numerically double integrate the raw X and Y, with the rotation component coming from the yaw of the NavX which is very accurate (see 5.3).

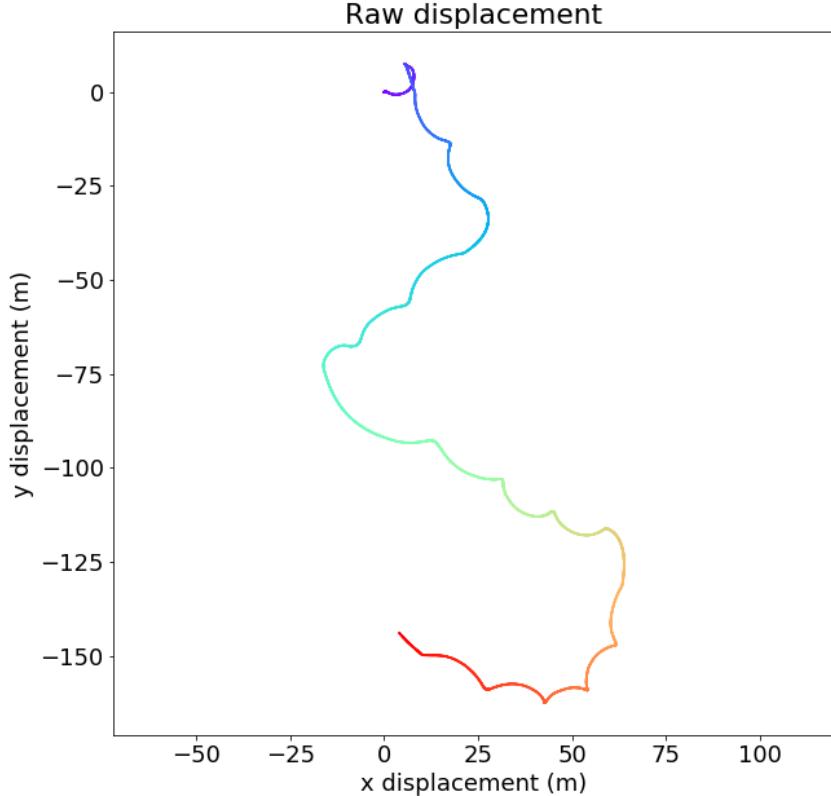


Figure 4: The plot shows position by double integrating raw accelerometer readings. Time proceeds from purple to red. The truth path was a set of 7 mostly concentric 4m diameter circles. After the first 1 seconds the data is inaccurate.

5.2 IMU Calibration

From an early experiment collecting data on a Turtlebot (section 5.1), we saw that double integrating the accelerometer readings was not accurate enough. This was expected, because it is well known that double integration will amplify any bias. Therefore, we replicated the IMU calibration procedure described in [39], which accounts for many sources of error without requiring expensive external equipment. This calibration method was straightforward to perform, and could be replicated by FRC students. This calibration method corrects the misalignment, scaling, and biases in both accelerometer and gyroscope. This is done by optimizing for accelerometer calibration values that make the magnitude of acceleration during static intervals closest to 1, and then by optimizing for gyroscope calibration values

that make the integral of gyroscope measurements between static intervals match the change in orientation between static positions. First, the IMU needed to be placed statically for a period of $T_{\text{init}} \approx 50$ seconds. Next, by calculating the variance of the accelerometer data collected during that initialization period, a threshold for a static interval detector could be determined by applying a constant multiplier. After the initial waiting period, the IMU needs to be rotated an arbitrary amount and left in that orientation for 1 to 4 seconds. Each IMU position during the “flip and wait” period should be distinct for calibration to be accurate. The entire “flip and wait” process has to be repeated 36 to 50 times. After all data was collected, an optimization procedure was ran first on the accelerometer data to solve for the calibration parameters for misalignment, scaling, and bias that make the norm of the acceleration closest to 1. Then, a similar method was used for gyroscope calibration based on the success of accelerometer calibration. The quality of calibration of gyroscope was entirely based on the quality of the accelerometer calibration.

In our experiments, we used $T_{\text{init}} = 50$, as was reported by the authors for a different IMU. The authors arrived at this number from a plot of Allen Variance—we did not reproduce this plot with our IMU. We waited 4s during our static intervals, but found that using $T_{\text{wait}} = 3$ was better in practice for detecting wide, clean, static intervals. This is possibly because sometimes the IMU was not truly at rest for a full four seconds. In our early experiments, we found that failing to record enough *distinct* static intervals would cause the optimization procedure to fail to converge. So, in order to get as many distinct positions as possible, a Helping-Hands was used to hold the IMU. We rotated the IMU 36 times in total, which was the minimum suggested number of static intervals in the original paper. The accelerometer data and gyroscope data in x , y , and z axis were recording for the entire period. Using the threshold from initialization data and the full accelerometer data, the static detector successfully distinguished between static intervals and dynamic intervals. A demonstration of our static detector is shown in Figure 5.

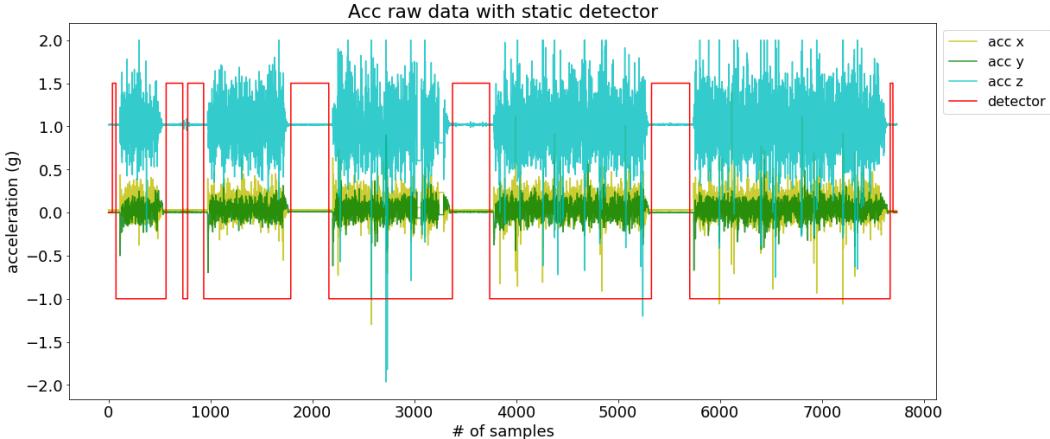


Figure 5: The black line is 1 during intervals classified as static

Using the identified static intervals, we optimize using the Levenburg-Marquedt procedure in python’s NumPy package to solve for the accelerometer calibration values. The equation we are minimizing is shown below (Equation 3). These values can be found in Table 1, and descriptions of each variable can be found in [39].

$$\|g\|^2 - \|T^a K^a (a^s + b^a)\|^2 \quad (3)$$

α_{yz}	α_{zy}	α_{zx}	s_x^a	s_x^a	s_z^a	b_x^a	b_y^a	b_z^a
-0.002710	0.004559	-0.000738	0.997279	0.996661	0.989960	-0.006376	-0.008999	-0.019918

Table 1: IMU Calibration Values

Note the values shown above are close to the values that represent no transformation, $[0, 0, 0, 1, 1, 1, 0, 0, 0]$. This indicates that our accelerometer is already quite well calibrated but not quite perfect, which is expected.

The next step is to calibrate the gyroscope. We integrate the angular rates measured by the gyro between every sequential pair of static intervals and compare this to the angle between the two static intervals. We have a good estimate of the true orientation of each static interval from the previous accelerometer calibration step, and so the goal is to solve for gyroscope calibration parameters that make the integral of the transformed gyroscope data over the dynamic interval match the next orientation of the static interval as measured from the calibrated accelerometer readings. This is expressed in the error function we are minimizing, shown in Equation 4.

$$\left\| u_{a,k} - \left(\int_{k-1}^k \Omega(\omega_i^S) di + u_{a,k-1} \right) \right\| \quad (4)$$

$$\Omega(\omega_i^S) = T^g K^g (\omega_i^S + b^g)$$

The function $\Omega(\omega_i^S)$ takes the raw angular velocity readings ω_i^S , transforms them with the calibration constants, and produces a rotation matrix. This rotation matrix is the euler rotation matrix (Roll-Pitch-Yaw ordering) which can then be multiplied by u_a . Towards this process, we investigated numerical methods for computing the above integral. This integral cannot be computed analytically because we only have samples of the integrand, rather than a analytic closed-form. Therefore, numerical integration methods like Euler's Forwardmethod or Runga-Kutta methods can be used. While [39] uses Runga-Kutta 4th Order (RK4), we used the 1-step Euler's Forward method. Over the whole integral, this rotates the average acceleration values from the $k-1$ th static interval, $u_{a,k-1}$, to the average acceleration values from the k th static interval. One could compute the same thing in a different order, by integrating the angular velocity values to get angles, constructing one rotation matrix, then rotating the acceleration values. However, because of gimble lock and dependence on ordering of the axis of rotation, this is much less accurate in practice. By rotating within the ingrand, we are only rotating by very small angles at a time, which mitigates the issues of using euler-angle rotation matrices. This theoretical result was tested experimentally, and the results are shown in Figure 6. Note that the bars representing the incremental rotation are more accurate than the one-shot rotation, where more-accurate is defined as closer to the true average acceleration readings at the next frame.



Figure 6: Integration of the gyroscope readings in the Y Axis. Method 1 is one-shot rotation, Method 2 is incremental rotation. Incremental rotation is clearly more accurate.

5.3 Accuracy of Gyro Integration versus On-Chip Yaw Calculation

We asked the question of whether the provided on-chip `GetYaw()` method is more or less accurate than what can be computed from the raw gyroscope readings. To answer this question, we first used implemented a simple procedure for computing yaw from the gyroscope readings. First, we apply the calibration parameters (see section 5.2), then a base-frame rotation. This base frame rotation accounts for the angle of mounting of the NavX on our robot, which may not be perfectly flat. To do this, we let the robot sit still for a second or two and compute the rotation matrix that rotates the accelerometers readings to be $[0, 0, 1]$, which is the value you'd expect if the NavX were flat. Having calibrated and rotated the raw gyroscope readings in all axis, we can consider only the yaw, or z axis, of the rotated data. We use a 1-step forward Euler's method to integrate these readings, which are in degrees/second. This gives us our yaw angle over time.

To compare this procedure with ground truth, we log the raw gyro values values while driving in the motion capture studio, then perform the calculations described above to get yaw. Figure 7 shows our computed yaw, compared with the on-chip `GetYaw()` and the yaw reported by motion capture. Due to the wrap-around behavior, the mocap yaw has a small blip in value that can be ignored. Overall, both our yaw value and `GetYaw()` match the ground truth very closely. The maximum error of 2.497° in the first 1000 samples (20 seconds).

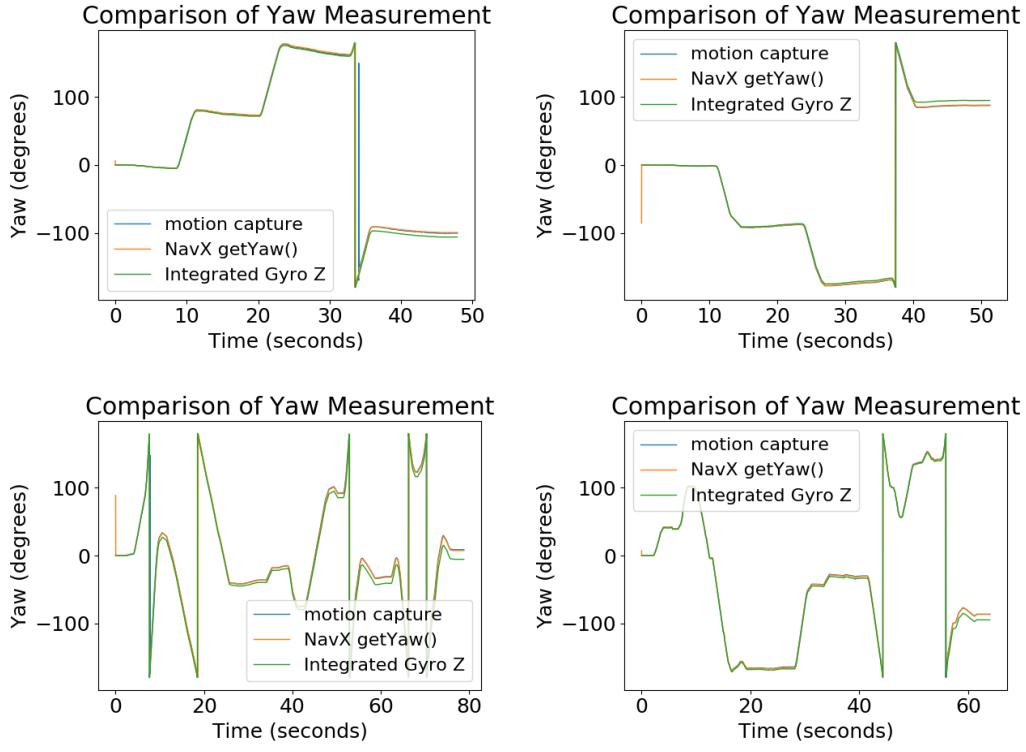


Figure 7: Comparison of yaw values between our algorithm and motion capture. The `GetYaw()` and Motion Capture lines are nearly indistinguishable.

Trial	Data Source	Average Error (deg)	90th Percentile Error (deg)
1	Navx GetYaw()	1.275	4.606
2	Navx GetYaw()	1.027	2.298
3	Navx GetYaw()	1.402	3.591
4	Navx GetYaw()	1.458	4.032
1	Integrated	3.619	7.710
2	Integrated	2.670	5.589
3	Integrated	6.315	13.659
4	Integrated	3.182	8.206

Table 2: Table of errors during 4 trials of the NavX on a Turtlebot under motion capture. The NavX is more accurate than integration and meets our criteria of accurate angle measurement (see section 4).

5.4 Characterising Drift and Bias in the Accelerometer

After confirming experimentally that integrating accelerometer readings would be inaccurate, we explore the well known techniques of drift compensation and zero velocity updates.

Before testing these directly, we first categorize just how much bias there is in our accelerometer, and how that bias changes over time.

5.4.1 Measuring the drift and bias in the accelerometer

Compensating for the accelerometer drift is important in IMU localization since double integration of accelerometer data amplifies any inaccuracies. We first performed an experiment to study the drift of accelerometer when it is stationary. We put the accelerometer on a flat surface for 6 minutes and collected the data at 100 Hz (see Figure 8). Then, we calculated the average accelerometer value of the first 500 samples and the last 500 samples in the x and y axes. The difference of mean values of first 500 samples and last 500 sample in x, y, z axes are -0.000475g, 0.000158g, 0.000323g respectively. For reference, we note that the maximum drift of -0.00475g, or $-0.00465 \text{ m s}^{-2}$ would cause a position error of $0.5 * -0.00465 * 3^2 = -0.020948 \text{ m}$ over a 3 second period. In other words, if the NavX is stationary, even if the initial bias of the accelerometer is zero, the position could drift up to 2 cm over 3 seconds.

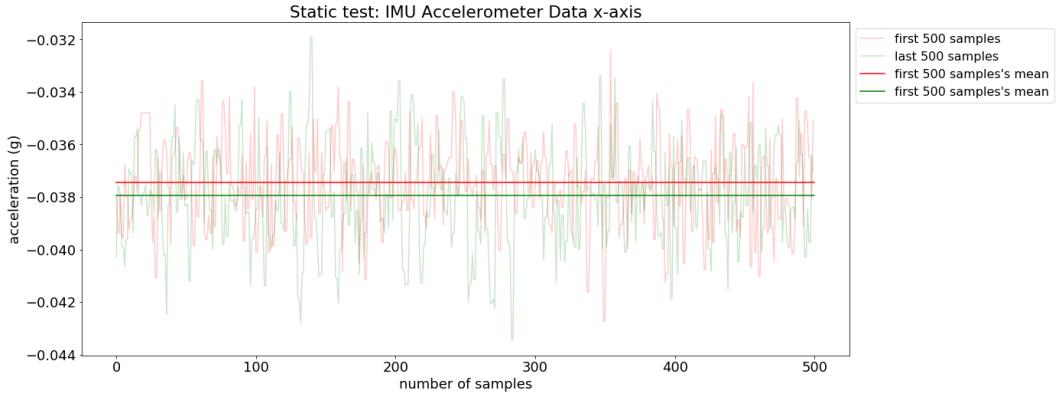


Figure 8: The raw measured X acceleration (Gs) and its mean over first and last 500 sample periods while stationary.

We then wondered that whether the duration of motion influences the amount of drift, so we performed another experiment. We drove the robot in a circle, stopped for 9 seconds, drove the robot in 2 circles, stopped for 9 second, so on until the robot drove for 5 circles in a row. We will refer to this test as the “Nypro Circles” test. This allows us to see whether moving for longer periods of times will cause more drift. We collected the accelerometer data, fused yaw measurement, and temperature. Using this data, we plot the mean accelerometer value in each of the static intervals to see if there is a clear trend (see Figure 9). Based on these means, we can say that the NavX accelerometer drifted a lot between static intervals. However, there is no simple linear trend between the duration of motion.

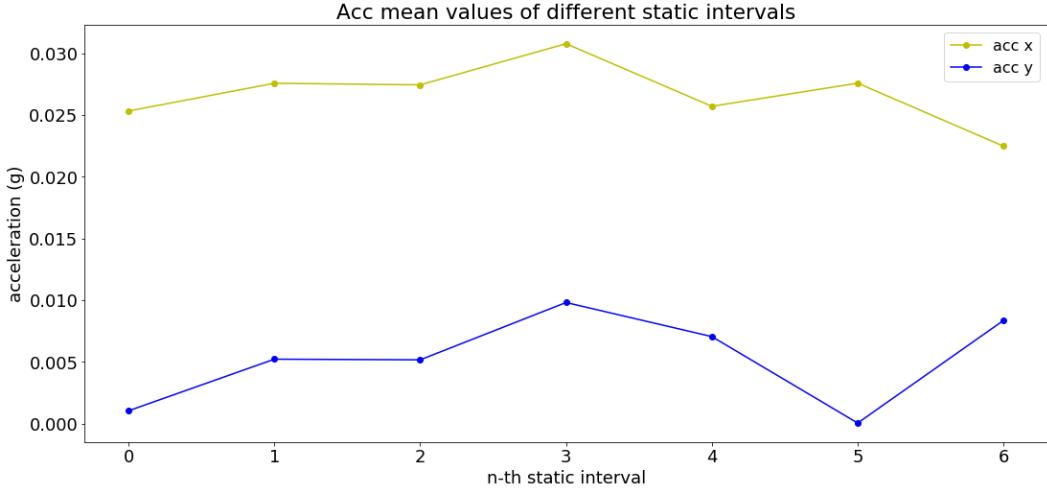


Figure 9: The means of the accelerometer data in world-frame X and Y in each static interval.

Having measured the accelerometer bias and studied its drift, we then integrated the accelerometer data with yaw angles of the “Nypro Circles” test to see how these effect the position. To get the best results possible, we also apply our calibration parameters (see 5.2). When integrating to get position, we rotate the robot into the world frame using the yaw angles come from the `GetYaw()` function of the NavX API, which is very accurate (see 5.3). Figure 10 and 11 show that bias and drift make velocity and displacement inaccurate after only a short period of motion.

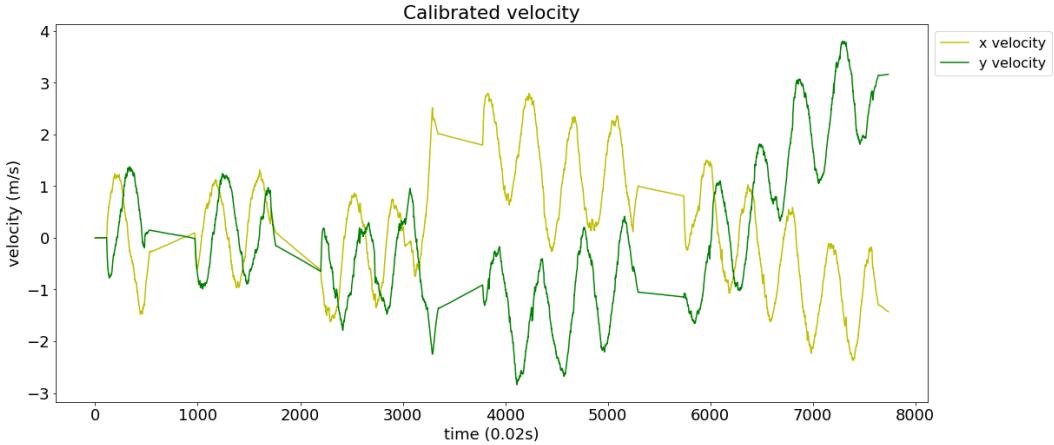


Figure 10: Velocity as derived by integrating the calibrated accelerometer measurements.

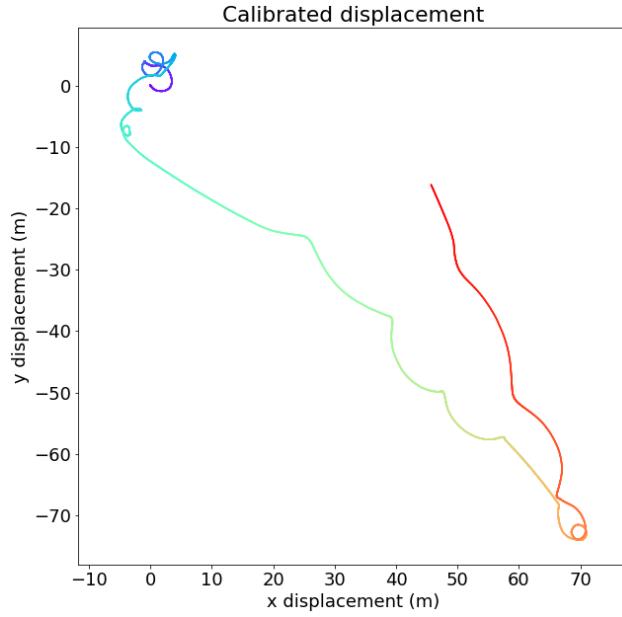


Figure 11: Displacement as derived by twice-integrating the calibrated accelerometer measurements.

Since temperature could also be a factor that affects accelerometer values, we compared the temperature with accelerometer values in static intervals over time. Shown in Figure 12, the temperature increased when the robot was static and decreased when the robot was in motion. However, temperature does not have a straightforward relationship with accelerometer bias or drift in bias.

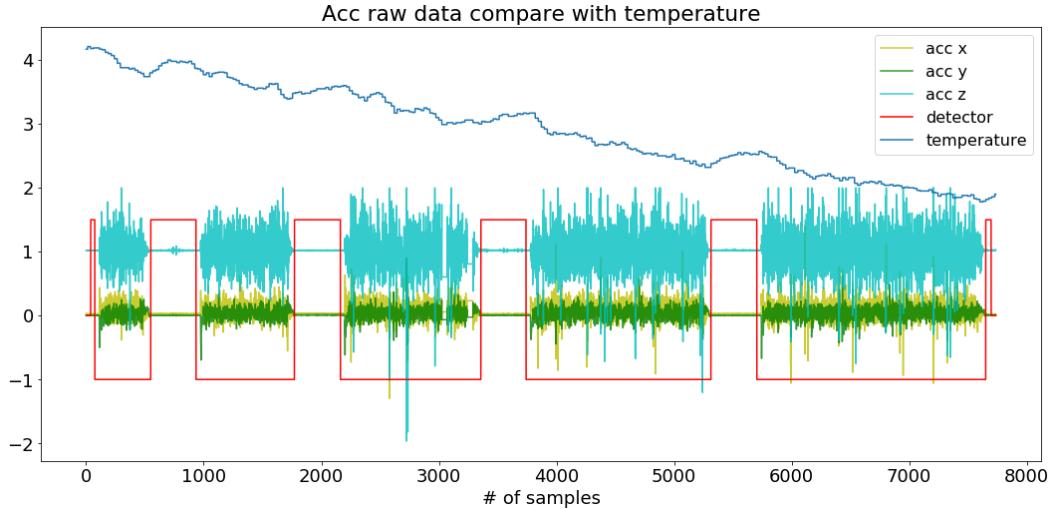


Figure 12: A Plot of temperature recorded by the NavX over the duration of our test.

Overall, our experiments showed that the accelerometer is subject to bias, and that these biases drift over periods of motion. Because of these errors, the double integration becomes inaccurate after a very short duration of motion. Furthermore, we show that the magnitude and direction of this drift has no straightforward relationship with the duration of motion or temperature. We now present several approaches for handling these sources of error and describe our results applying them to this data.

5.4.2 Zero Velocity Updates

Looking at the calibrated velocity plots (Figure 10), clearly there is still bias in the accelerometer readings which are causing the velocity to drift up and down during intervals of motion. We now apply zero velocity updates to the data to mitigate this. Our zero velocity updates work as such: when the static detector indicates the robot is stationary, calculate the bias in that static interval, remove that bias from the static interval and the following dynamic interval, and finally set the current velocity estimate to be zero.

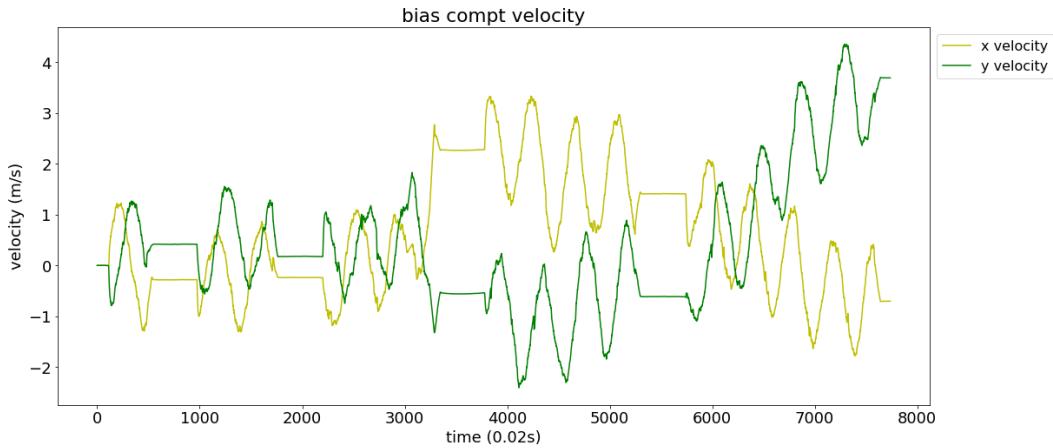


Figure 13: Velocity after bias during static intervals is removed.

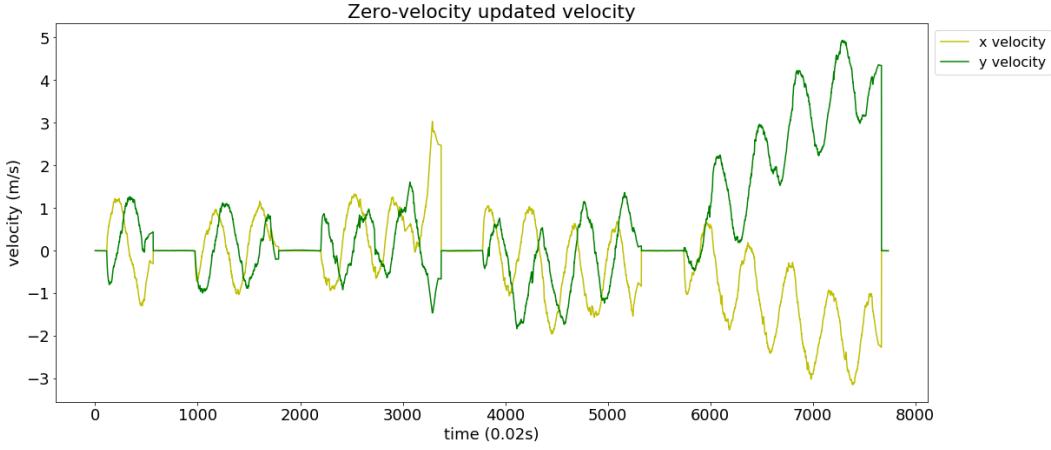


Figure 14: Velocity after both applying bias and zeroing velocity estimates.

5.4.3 Drift Compensation

Having compensated for the biases found during static intervals, we now wonder whether we can account for the drift between these static interval biases. As was shown in Section 5.4.1, there is no clear trend in how the motions change between or within static intervals. This means we cannot hope to find any one drift rate and apply it to the entire stream of accelerometer data. However, there are two other possible ways to account for drift looking only at a single static interval or a single pair of static intervals. The first method is to calculate the drift rates *between* two sequential static intervals, and apply drift compensation starting from the static interval and up until the next static interval. This method is an offline method since it requires knowing the future accelerometer data to account for the drift in the current data. Because this method requires future data, it is not possible to implement in our system in real time. The second method is to calculate the drift rate *within* a static interval and project this drifting behavior on both the static interval and the following dynamic interval. This method is online because it only requires current and past accelerometer readings. Both of these methods offer no significant improvement, but we report them for completeness. These two methods are plotted below, with the original data (only calibration applied, no drift compensation) shown for comparison (Figures 15, 16, 17).

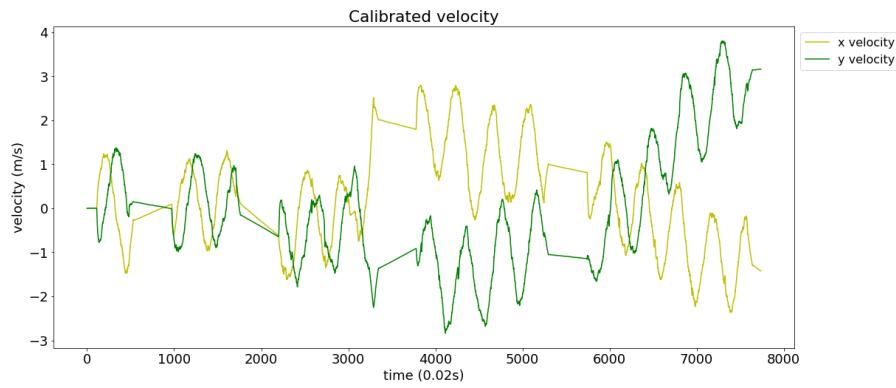


Figure 15: Velocity as derived by integrating the calibrated accelerometer measurements.

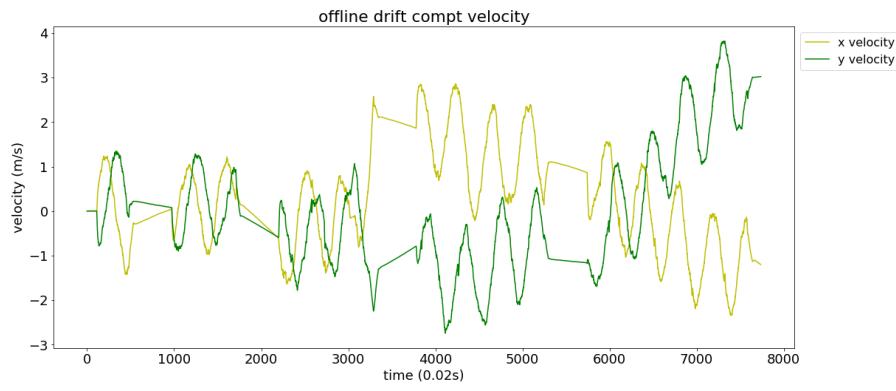


Figure 16: Velocity where drift is calculated between static intervals

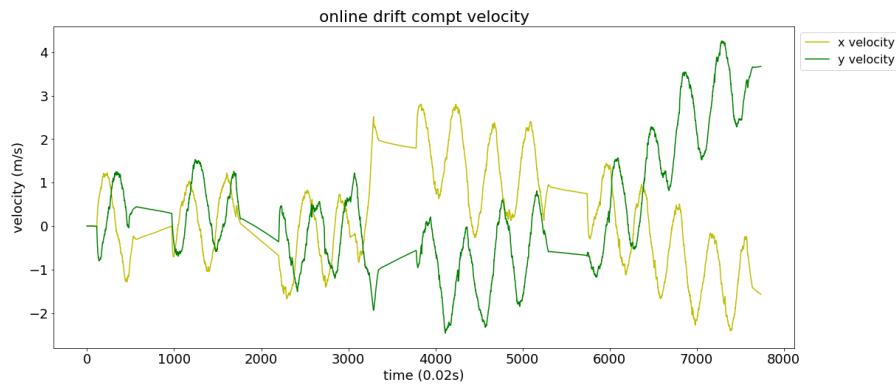


Figure 17: Velocity where drift is calculated within static intervals

5.5 Comparing Our IMU Localization to the NavX API

We compare our method for localization with only the IMU against the `GetWorldX()` and `GetWorldY()` functions provided by the NavX. These functions differ from our methods because they do not include zero velocity updates or drift compensation. We also apply our own calibration parameters to our data, which differs from the internal calibration done by the NavX. As is shown in the Figure 18, both methods drift significantly over the course of our experiment (“Nupro Circles”, described in Section 5.4.1). For reference, in this test our robot was driven in circles with a constant left-right wheel speed difference. However, when we zoom in to the first thirty seconds of the data, we see our method better preserves the sinusoidal nature of the motion, whereas the NavX position constructs lots of straight edges. Furthermore, zooming in to the first three seconds highlights that our method is significantly more accurate.

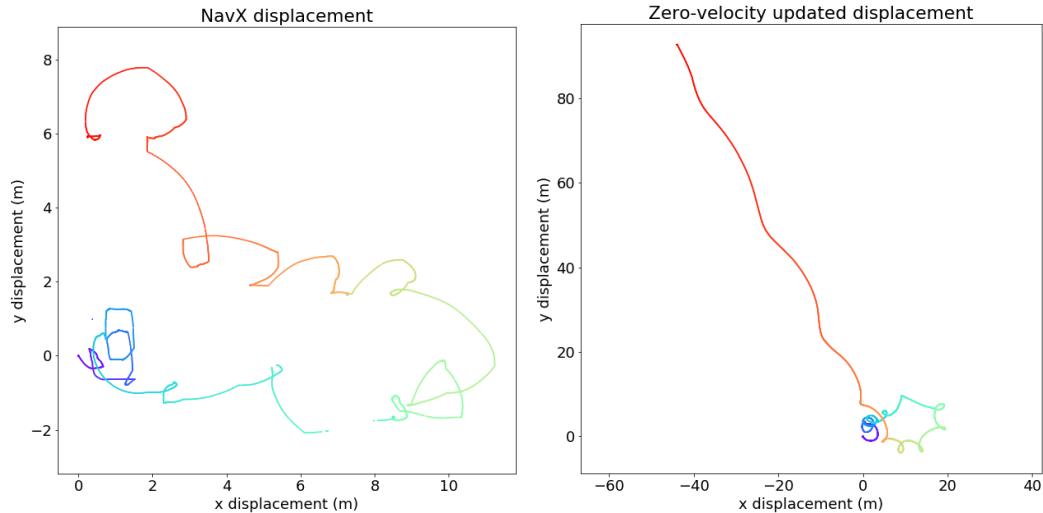


Figure 18: Comparison between NavX (left) and our method (right) over the entire experiment.

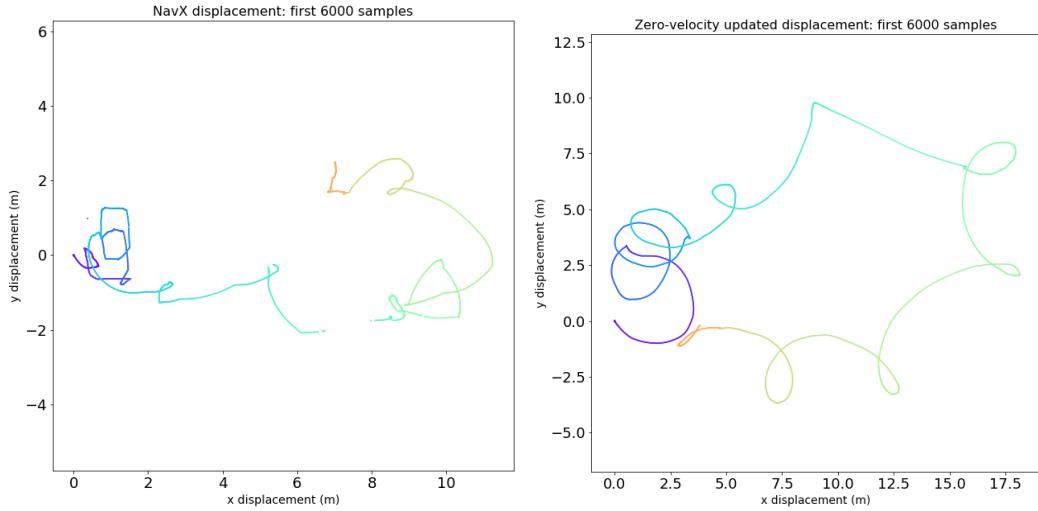


Figure 19: Comparison between NavX (left) and our method (right) over the first 30 seconds of the experiment.

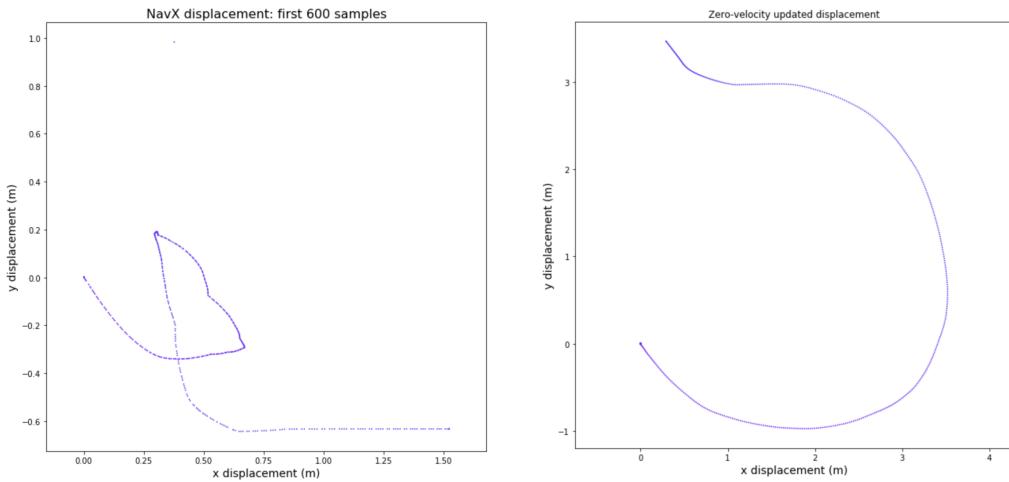


Figure 20: Comparison between NavX (left) and our method (right) over the first 3 seconds of the experiment.

5.6 Measuring Beacon Delays

The beacon system relies on measuring the time it takes for a sound signal to travel from the beacons to the robot. To do this accurately, one must account for transmit and receive delays in addition to the actual time of flight. Figure 21 illustrates the various delays we need to account for. We conducted experiments to estimate these delays.

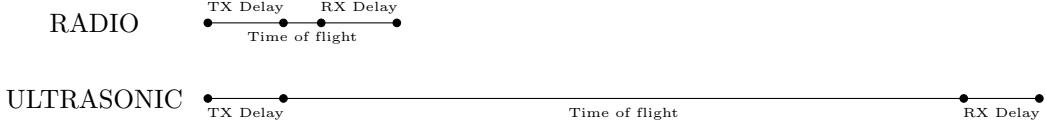


Figure 21: Timing of radio and ultrasonic signals. Experiments indicate 46.175 μ s total RF delay and 1 ms total ultrasonic delay.

First, to get an estimate of the radio transmit and receive delay, a transmitter and receiver were set up on two microcontrollers. The transmitter sent 5 ms pulses at 433 MHz (no encoded data) every 55 ms, and oscilloscope probes were attached to the input pin on the transmitter and the output pin on the receiver. By comparing the time difference between the input and output signals on the oscilloscope, we can determine the total time. Furthermore, we can measure the distance between the transmitter and receiver and subtract the theoretical time of flight from the total time. The full data for these measurements are available in *Radio Time of Flight*, and an example measurement is shown in Figure 22. The time of flight of radio over distances of a few centimeters or meters is on the order of nanoseconds. We measured an average delay of 45.175 μ s, which we attribute to the internal circuitry of the transmitter and receiver. The variance of this delay was 16 μ s. However, we also measured delays as low as 32 μ s and as high as 79 μ s. Since the theoretical time of flight over the distances used in this experiment were at most 1 ns, we can conclude that there is both delay and significant variance in the delay of the transmitters and receivers. This is an important delay to consider when implementing the timing measurement of the beacon signals.

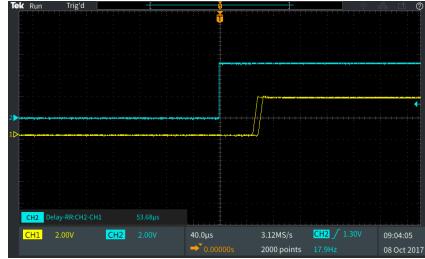


Figure 22: Example measurement total trip time for radio signal. The blue line is the input to the transmitter, and the yellow are the output of the receiver

Next we performed a similar experiment with the ultrasonic transducers. For this experiment, we used two NTX-1004PZ piezo speakers placed 25 cm apart. The NTX-1004PZ is meant to be a high-frequency speaker for DJ equipment, and is designed to operate between 4 kHz and 20 kHz. However, because they are incredibly cheap we decided to evaluate them as ultrasonic speakers running just above that range. One was connected to a PSoC 5LP for transmitting, and the other was connected only to the oscilloscope. The other oscilloscope probe was connected to the transmitting piezo. The time difference between the transmitting signal and the receiving signal was measured. The signal applied to the transmitter was short bursts of a 24 Hz square wave. Again, the distance was measured between the transmitted and received waveform, and the theoretical time of flight was subtracted. The full data for this experiment is shown in table 3.

Distance (m)	Expected Delay (us)	Measured Delay (us)	Error (Measured - Expected)
0.10	294	390	96
0.15	441	556	115
0.20	588	698	110
0.25	735	872	137
0.30	882	1001	119

Table 3: Measured Delays in 2kHz Sine Wave Signal

This data suggests that there is a constant delay of ≈ 115 s, which could be attributed to the internal amplification circuitry and the time for the receiving piezo to begin to resonate. An example of the oscilloscope readings is shown in Figure 23, which illustrates the time period where the receiving piezo response is building up before becoming detectable.



Figure 23: Capture of the measurement of ultrasonic delay on the oscilloscope

5.7 Measuring Frequency Response

After testing for delays, we also measured the frequency response of the NTX-1004PZ piezo speaker. We placed two speakers 17 feet apart, and using a function generator we transmitted a square wave at 8vPP and swept from 20 kHz to 30 kHz and back down over the course of 20 seconds. We attached an oscilloscope to the receiving speaker and captured the power at each frequency using the FFT mode, persisting the display over the course of the sweep to see how the frequency response changes across our frequency range. Figure 24 shows the results of this experiment. From this experiment, we learned that the best frequency response is achieved at 22 kHz, and after 27 kHz the signal is indistinguishable from the noise.

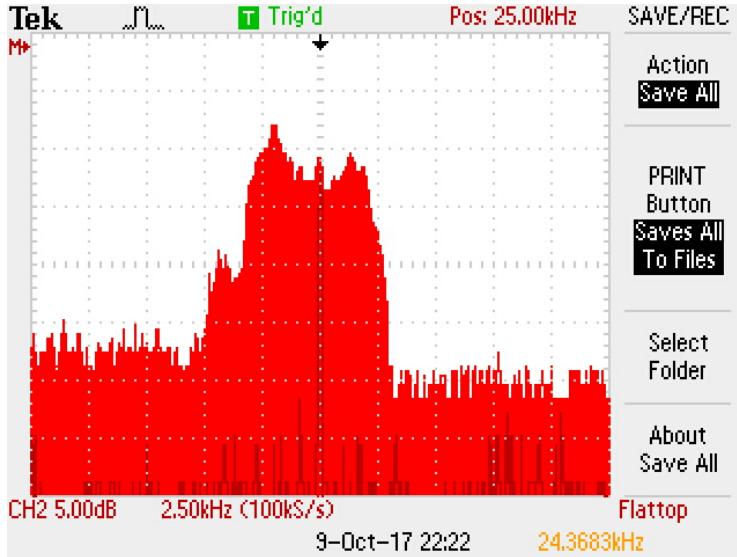


Figure 24: Frequency response of the NTX-1004PZ, centered at 25 kHz with 2.5 kHz per division. The best response is achieved at 23 kHz, and the highest detectable frequency is 27.5 kHz.

This experiment shows that any ultrasonic signals emitted by the beacons must be within the 20-27kHz range. For fixed frequency signals, 22kHz should be used. Lower frequencies will be detectable and painful or annoying to humans, and higher frequencies will be undetectable.

5.8 A Theoretical Procedure for Building a Map of Beacons

In order to use beacons to localize, the absolute positions of the beacons must be known. Naively, one could simply place the beacons in fixed locations and measure the position with respect to the field or practice space. However, this is an unsatisfactory solution for our use case in high-speed multi-robot gameplay. It is inevitable that collisions with robots or people working in the space will bump the beacons and change their position. Furthermore, we found in our survey that some FRC teams use a classroom as their practice space, and therefore are unable to leave beacons out in the same position for extended periods of time. Therefore, we describe a procedure by which the beacons, upon initial setup, can discover their own relative positions.

Consider a “Cricket” style beacon using radio and ultrasonic communication like those described in section 2.4. Because each beacon is equipped with radio transmitter receiver pair and a piezo transducer, any beacon can send and receive radio signals or ultrasonic chirps to or from any other beacon. This is the principle we will use to construct a map of beacons. The mapping procedure occurs upon startup of the system, or possibly periodically whenever the user believes a new map should be built. We also designate a “Master” beacon, which is simply the first beacon that is turned on. The list below outlines the steps required:

1. Identification

- (a) Turn first beacon on, which becomes the master
- (b) The master will begin to broadcast itself with a radio message
- (c) Turn each other beacon on. Each beacon will hear the master's broadcast message and broadcast a request a Id assignment
- (d) The master will hand out sequential Ids to each beacon
- (e) After all the beacons have been assign, the identification stage is complete

2. Range Data Collection

- (a) The leader starts emitting orders to beacons to send ultrasonic (US) signals to locate the other beacons
- (b) When beacon hears its signal, it will chirp US
- (c) Everyone else will listen for that US and compute their distance to beacon 1
- (d) Then beacon two will hear its signal, and will chirp US
- (e) Everyone else will listen and compute distance to beacon 2
- (f) Repeat for all the identified beacons

3. Map Construction

- (a) At this point, all of the beacons have computed all of the ranges to all other beacons
- (b) The master will then one-by-one request each beacon to emit this information
- (c) Once the master has collected all range estimates, it uses a least-squares solver to find the distances that minimize the error from all the range estimates

The final step in this procedure is a simple optimization step. The problem can be stated formally as such. Let there be N beacons, let d_{ij} be the true distance from beacon i to j , and let \hat{d}_{ij}^k by the distance from i to j as measured by beacon k . The optimization problem is as follows:

$$\arg \min_{d_{ij}} \sum_{k=0}^N \|d_{ij} - \hat{d}_{ij}^k\|^2 \quad (5)$$

Because we formulate the optimization problem as a sum of square error, there are many potential optimization methods that could be used, such as Levenburg-Marquedt. The end result will be a set of distances from each beacon to each other beacon. From this point, one can either assume that a given beacon (sensibly beacon 0) is the origin, or one can provide the position of the origin beacon with respect to some other origin on the field of practice space. Either way, this setup procedure and optimization problem result in a map which can be used to find the position of the robot give any collection of measured ranges to three or more beacons.

5.9 OpenCV Optical Flow Sample Code

Preliminary testing with optical flow was done using a Microsoft USB camera using the sample code provided in OpenCV. In the screenshot below the window labeled flow that there are a variety of green dots on the screen. These are the points that dense optical flow has identified. There is also a green line which is the motion vector of which way the frames are moving. The middle window labeled HSV flow is adding color to the different points that are currently the best for tracking on the frame. The bottom window labeled glitch is the current frame and previous ones overlaid showing all of the motion that has happened.



Figure 25: Screenshot of the opencv sample program `1k_track.py` on video collected on a practice FRC field. Aruco tags provide excellent targets for Lucas-Kanade tracking.

5.10 Benchmarking OpenCV Processing Times

This test compares computation time for optical flow with OpenCV. Tests were done using `lkdemo.cpp` which was we modified from a sample file provided by OpenCV. We compare this program on a laptop verse the RoboRIO and compare the time they took to run the code. The laptop used has a 2.8 GHz Intel 4 Core i7 processor. A chart below was made of the time that each program took to run 100 frames in seconds.

	Laptop (sec)	RoboRIO (sec)
3.638	8.429	
4.184	8.429	
3.638	8.429	
3.639	8.429	
4.184	8.429	
Average (sec)	3.8566	8.429
Average (FPS)	26	12

Table 4: Time for 100 frames to run using OpenCV on laptop verse RoboRIO

We performed these measurements 5 times to ensure repeatability. From these numbers, we conclude the laptop was just over twice as fast that of the RoboRIO. Based on our results from section 5.18, we conclude that 12 FPS is not fast enough for our project requirements and so a co-processor is needed.

5.11 Collecting Ground-Truth with VICON Motion Capture

To evaluate the accuracy of our system and to help with tuning various constants in the system we need a source of ground-truth state information. The ground truth data for measuring accuracy and precision is obtained using a VICON brand Motion Capture system. This comprises a VICON Lock+ data processor and 8 Vero infrared cameras. Our system can collect 2.2 megapixels of data and is designed for capturing human motion in small spaces. The VICON system is accurate to approximately 1 mm. In our experiments, the space used for experimentation was 19x14 feet. The pose of the robot is tracked using three retro-reflective markers. These are positioned at known distances such that the transform between the centroid of the markers and the centroid of the robot is easily obtained. A scalene triangle laser cut from acrylic was used as a guide.

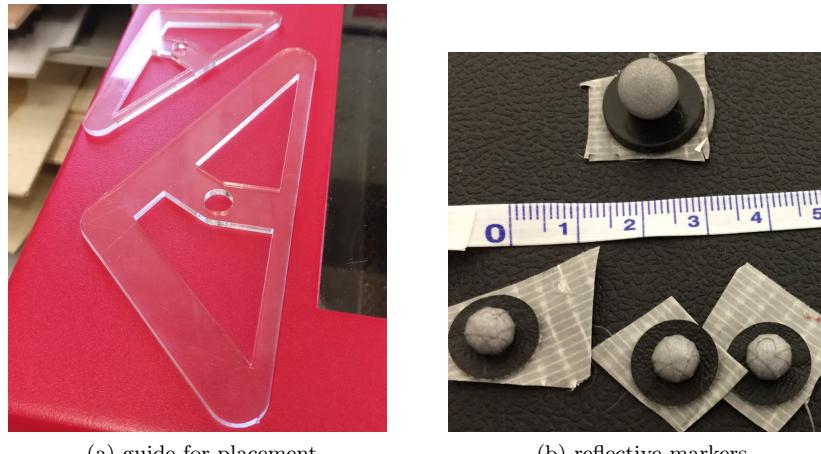
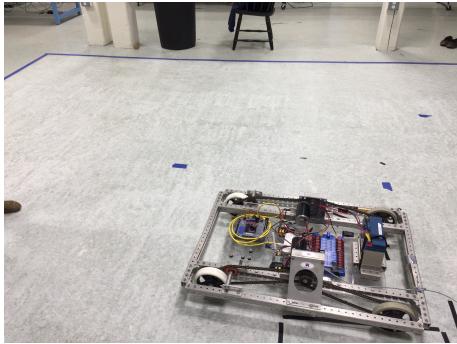
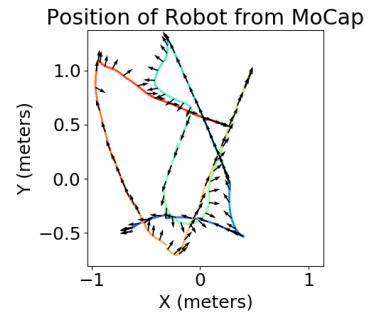


Figure 26: VICON tracker set up

In our experiments, the camera system captures data at 100Hz. To synchronize data collection, the RoboRIO sends a 5V signal to the Lock+ processor, and a UDP packet is transmitted to the Co-Processor running the camera. This data is synchronous to within ≈ 500 μ s. Using the same markers, the pose of the ArUco tags is also measured.



(a) robot in VICON field



(b) VICON (blue to red over time) position and orientation data

Figure 27: Collecting and Plotting Position data

5.12 Detecting Simulated Chirps in MATLAB

In order to examine the theoretical limits of our ultrasonic chirp detection, we created synthetic chirps and examine how pattern matching filters would work to detect them. For our beacons to work we must be able to very precisely find the start of a chirp given a buffer of ADC readings, and we simulate this in MATLAB. In these experiments, we construct our chirps using matlab's `chirp` function, and we sweep from 20-27kHz (see section 5.7 for justification). This signal is shown in figure 28. The zoomed in version highlights that given a reasonable ADC speed of 108ksps, we will only see a very rough sine wave.