

# Localization for FRC

## C Term Report

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

March 5, 2018

## 1 Abstract

## 2 Introduction

Knowing the position and orientation of a mobile robot is critical to many tasks. For robots designed for high-speed gameplay, knowing the position and orientation allows the robot to perform complex autonomous behaviors such as shooting and retrieving game objects. In this report, we describe a system for determining the pose  $(x, y, \theta)$  of a mobile robot in a cluttered environment. We are using the use case of FIRST Robotics Competitions (FRC) as the environment for testing the platform. The localization platform is being designed as an all purpose localization platform with focus on the FRC environment due to the interesting challenges FRC provides. 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, as well as 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.

At the highest level, our project is to allow FRC robots to query their position on the field. This information is a prerequisite for interesting behaviors such as path following, picking up game pieces, navigating to feeder stations, and detecting slips and collisions. These behaviors are currently beyond the reach of most teams, so we hope that by providing a system for localization, teams can extend our work to these complex behaviors. An important criteria for our system is that it work well not only on the FRC field, but in teams practice spaces. If teams are not confident that their system will perform in competition the same way it performs in their practice spaces, they will not use. The risk is simply too great. This is a challenge that most previous localization systems do not consider. Even though FRC teams do not have a full field to test on, they still need a way to test specific behaviors that are specific to the field. Presently, teams will construct mock-ups of small parts of the

---

\*equal contribution from all authors

field and various game elements to test with. For example, if the game requires shooting basketballs into a hoop, most teams will build or purchase a basketball hoop that is similar to the one used on the field and mount it somewhere in their shop. The important point is that the hoop will likely be mounted in a slightly different position or orientation. Ideally, using our system teams would be able to use their mock setup as if it were the real field. In other words, teams could test a program that autonomously shoots balls into the hoop in their shop and later use the same code with confidence in the competition. Teams must be able to specify the location of their mock field elements so that our localization system can report position that is relative to this hallucinated version of the field. This report begins with a review of existing literature on indoor localization for mobile robots in the Survey of Localization Techniques section. The strengths and weaknesses of these existing methods are described in the Trade-Off Analysis section. The Experimental Results section contains information on experiments we conducted. The System Specification section provides a complete description of our design criteria and system specifications. The Conclusion section details our plan moving forward.

## Contents

<b>1 Abstract</b>	<b>1</b>
<b>2 Introduction</b>	<b>1</b>
<b>3 Survey of Localization Techniques</b>	<b>3</b>
3.1 LIDAR Mapping . . . . .	4
3.2 Ultrasonic Mapping . . . . .	5
3.3 IMUs and Encoders . . . . .	5
3.4 Beacon systems and Wireless Networks . . . . .	7
3.5 Cameras with Visual Tags . . . . .	8
3.6 Optical Flow . . . . .	8
3.7 Filtering and Calibration . . . . .	10
<b>4 Trade-Off Analysis</b>	<b>11</b>
<b>5 Experimental Results</b>	<b>14</b>
5.1 IMU Calibration . . . . .	14
5.2 Accuracy of Gyro Integration versus On-Chip Yaw Calculation . . . . .	17
5.3 Characterising Drift and Bias in the Accelerometer . . . . .	18
5.3.1 Measuring the drift and bias in the accelerometer . . . . .	18
5.3.2 Zero Velocity Updates . . . . .	21
5.3.3 Drift Compensation . . . . .	22
5.4 Comparing Our IMU Localization to the NavX API . . . . .	24
5.5 Measuring Beacon Delays . . . . .	25
5.6 Measuring Frequency Response . . . . .	27
5.7 OpenCV Optical Flow Sample Code . . . . .	28
5.8 Benchmarking OpenCV Processing Times . . . . .	29
5.9 Detecting ArUco Tags at a Distance . . . . .	29
5.10 Collecting Ground-Truth with VICON Motion Capture . . . . .	29
5.11 Localizing with IMU and Encoder on the Turtlebot . . . . .	31

5.12	Detecting Simulated Chirps in MATLAB . . . . .	31
5.12.1	The Doppler Effect on Ultrasonic . . . . .	32
5.12.2	Effect of Chirp Bandwidth . . . . .	32
5.13	Detecting Ultrasonic Signals on the PSoC 5LP . . . . .	33
5.14	Ultrasonic Beam Angle & Full Pattern . . . . .	33
5.15	Characteristics of Piezo Transducers . . . . .	33
5.16	Co-Processors for Image Processing . . . . .	33
5.17	Evaluating The Placement of ArUco Tags . . . . .	34
5.18	Statistics of CSCore Image Timestamps . . . . .	35
5.19	Effect of Frame Rate and Resolution on ArUco Tag Detection . . . . .	37
5.20	Rate of position estimates from ArUco Tags . . . . .	38
5.21	Benchmarking MarkerMapper with VICON Motion Capture . . . . .	39
5.22	On Building MarkerMaps . . . . .	39
5.23	Erroneous detections with ArUco . . . . .	39
5.24	Latency over the Robot Network . . . . .	40
<b>6</b>	<b>System Specification</b>	<b>41</b>
6.1	Design Criteria . . . . .	41
6.1.1	ArUco SLAM . . . . .	43
6.2	Kalman Filtering . . . . .	44
6.3	System Requirements . . . . .	44
6.4	Timeline and Goals . . . . .	45
6.5	Implementation Details . . . . .	46
6.5.1	The Test Robot . . . . .	46
6.5.2	General Architecture . . . . .	46
6.5.3	NavX Processing . . . . .	47
6.5.4	Encoder IMU Fusion . . . . .	48
6.5.5	Radio and Ultrasonic Beacons . . . . .	49
6.5.6	User Interface (GUI) . . . . .	51
<b>7</b>	<b>Conclusion</b>	<b>51</b>
<b>8</b>	<b>Acknowledgements</b>	<b>52</b>
<b>9</b>	<b>Appendix</b>	<b>52</b>
9.1	Appendix A . . . . .	52
9.2	Appendix B . . . . .	53
9.3	Appendix C . . . . .	53

### 3 Survey of Localization Techniques

Overall, the problem of localizing a mobile robot can be viewed as accurately measure 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 points of view, 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 time of arrival to calculate distance 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, ultrasonic, cameras, inertial measurement units (IMU), encoders, radio, infrared light, visible light, and human-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:

- 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 in order to evaluate them. 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.

### 3.1 LIDAR Mapping

LIDAR is a sensor that works by measuring the amount of time it takes light to return to the LIDAR after hitting a desired object [20]. Light travels at about  $0.3 \text{ m ns}^{-1}$  which is a constant speed [20]. The sensor sends a certain number of beams at certain intervals towards an object. Since light moves at a constant speed the LIDAR can calculate the distance between itself and the object that light was hitting. To calculate distance the formula  $\frac{d*c}{2}$  can be used.  $d$  is the distance to the object,  $c$  is speed of light and then divided by two to account 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 [20].

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 [20]. 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 [20]. This project would use a topographical LIDAR to measure the distance from itself to other objects to find position.

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 [20]. The wavelength for these is typically 1.0-1.5 m [46]. The second system uses high energy lasers and is typically only used for atmospheric measurements [20]. The wavelength of these is typically 0.5-0.55 m [46]. LIDAR does localization by using the map it generated of its surrounding and being able to identify landmarks on the map. The LIDAR system will then compare the location of the landmarks to a previously known map. And since the distance between it and those landmarks are known, the LIDAR system can be used to determine its own position [39]. 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 [25].

### 3.2 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 [10]. Alternately, some systems use one sensor and rotate it around to achieve the same effect [24, 10]. The algorithms for interpreting the measured distances work by first extracting lines, then matching these lines to an existing map. Reported accuracies of the system in [10] was 1 ft for position, and 10° for angle. In [10] and [24], 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 [42], 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.

### 3.3 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, 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) [2]. 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 [32]. High cost IMU's 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 [2].

In cost-sensitive systems, this methodology is subject to high error rates from accumulation. Because of integration of accelerometer data, the velocity error term grows linearly and position error quadratically. This introduces a need for filtering, sensor fusion, and optimization based approaches.

Rarely in mobile applications are IMUs the primary position sensor. Odometers and encoders offer relative position sensing because they measure change between distances, but must be provided with a frame of reference. Most odometry-based sensors are updated at

a high-frequency, but are subject to high error rates from gear inefficiencies, wheel slippage during normal rotation, and irregularities in data processing [14]. Complementary filtering, or using multiple, weighted sources of data, is used to obtain a position estimate. Linear quadratic estimation can be used to estimate the position of a object; however, this technique is relatively complex. The algorithm makes a prediction about the current position, taking into account error from the previous measurements. Once the current data is available, the algorithm uses it to correct the parameters it used to make the estimation, provided the data is of high certainty. Known as a Kalman filter, this process uses a known model of the system and assumes errors in the sensors to smoothen the data. Systems can leverage data from a range sensor [44] or indoor positioning systems that use radio frequency signals [29].

Cameras, radio beacons, GPS, and similar landmark-based technologies offer global position sensing and have lower accumulated error than local systems discussed above. However, update frequencies are comparatively low, leading to false predictions or periods of unknown position. To compensate for this phenomenon, systems can leverage data streams with higher update frequencies to interpolate between frames. Forster *et al.* describe a visual-inertial navigation system that uses accelerometer and gyroscope data streamed at a high frequency to estimate a pose trajectory between select frames from the camera. Known as keyframes, these camera data are selected to minimize computational requirements while minimizing feature loss. Often, a parallel thread selects appropriate frames. Then, the IMU data are processed into a relative motion constraint. Effectively, this yields a trajectory of position and orientation between camera updates [4].

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. Known as preintegration, this technique, instead of filtering the data, 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 decreased the computational load of the localization thread [28]. Systems like this are effective because camera data is used to correct IMU data on-line when data is available and rarely fails to update the pose estimate between frames. The above work describes an overall CPU time of about 10ms for data processing and real-time execution, although the system update frequency is unknown.

Leveraging preintegration, systems expanded sensor fusion frameworks with probabilistic terms and models of relationships between sensors. One such approach used a factor-graph to represent system state variables as nodes on a tree and the functional relationship between them as factors (edges). Instead of updating the state each time new data is available, the factor tree updates relevant state variables add into account an error term [47]. A key aspect of this system relies on frame transformations between sensor data happening at a low level. This takes workload off of the main CPU by utilizing FPGAs or other processors to monitor and process odometry or IMU data. Implementations of such systems claim reduced computational load and similar performance to ORB-SLAM and other modern navigation systems.

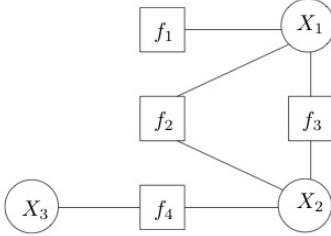


Figure 1: A simple factor graph [16].

Designed for the FIRST Robotics Challenge, frameworks such as Sensor Fusion 2 (SF2) provide users with algorithms for an integrated platform. Some examples are latency correction between IMU and camera data, fusion of encoder and IMU data, and keyframe-based state estimation [17]. Again, 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. Usually, the IMU data is available at a rate an order of magnitude larger than that of the camera. 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. This is useful for trajectory planning or on-line path correction (like recovering from a collision). Within the next several years, systems like this will mature, offering localization or possibly SLAM.

### 3.4 Beacon systems and Wireless Networks

Beacon systems have been used many times with success in the literature. Generally, these systems 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 5m<sup>2</sup>. Another RFID system [37] 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 [27]. 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 [52] [35].

Among ultrasonic beacon systems, [22] 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 [40], [49], and [21] send radio pulses followed by ultrasonic pulses. 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 [13] [51]. These systems are inexpensive, and report accuracy of between 2 cm and 14 cm.

### 3.5 Cameras with Visual Tags

Vision based localization is widely used in robotics. Different visual approaches have been applied to mobile robot systems. Most of vision based systems have cameras on the robots and had either natural landmarks or artificial landmarks in the environments as references to absolute positions. If the natural landmark approach is used, knowing the absolute positions of objects in the circumstance, a robot could calculate its absolute position by calculating orientation and distance toward the object detected and recognized with its absolute position. Another similar approach is using 3D models and 2D to 3D matching techniques. The system described in [38] had accurately localized the camera's position using this 2D to 3D mapping technique. Among all of the vision based approaches, the usage of a combination of cameras and artificial landmarks to localize robots is one of the most popular. Common artificial landmarks include 1D binary barcode, 2D binary barcode and 2D colored barcode. The system in [26] applied the approach of using cameras and ID tags. The system had ID tags on the ceiling, which were 2 m away from the floor. A web camera, facing to the ceiling, was mounted on a moving robot with a speed of  $20 \text{ cm s}^{-1}$ . This system measured the position and orientation of the robot relative to the ID tags through image processing. The result of the experiment in [26] showed that this method was accurate even though there was an unevenness between the ceiling and the floor. Another system [15] 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, it only had few centimeters of error. Another barcode based localization system of low capacity mobile robot (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  $3.5 \text{ m s}^{-1}$  of error on average. An experiment on 2D barcode based localization was performed on a Pioneer 3 - AT robot. The robot was placed in a random location in the environment and had a Canon VC-C50i analog camera connected with a 1.8 GHz processor and worked under Linux operating System. The robot moved with a speed of  $50 \text{ mm s}^{-1}$ , trying to locate itself by finding a code vertically mounted on a wall. The code contained information about its xyz position, normal vectors and its length, therefore, the system would not need to query through databases to find information, which made the process faster.

### 3.6 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 where given a sequence of images it can find the movement of objects between the images. More exactly 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 calculate in order to calculate 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 making it considered a global method. 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[33].

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 gradients refer 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 [41].  $\alpha$  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 2 [33].  $n$  represents which iteration the optical flow calculation is on. Each current pixels optical flow is calculated based on previous iterations pixels. Optical flow calculation will iterate from pixel to pixel until it has calculated optical flow for each pixel.

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

Figure 2: Horn and Schunk Optical Flow vector equation

Optical flow can also be done locally using the Lucas Kanade method [41]. 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 3 [33].

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

Figure 3: Lucas Kanade Optical Flow vector equation

$\nabla I(p)$  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 [33].

Lucas Kanade looks at pixels which have a large gradient in intensity of color. It changes

images to be gray scaled in order to see changes in darkness instead of colors. Lucas Kanade method works the best when it uses pixels that have three qualities. The first is that the pixels in a small area have the same brightness. Even though the location of the pixel has changed the brightness is consistent, so using a pixel from that area to track is optimal. They also should be temporally similar meaning the path the pixel takes should change gradually. Pixels should also be spatially similar, meaning that neighboring pixels should have similar motion to each other. Picking pixels that fit these requirements work best with Lucas Kanade [41].

Optical flow has been used for multi-sensor localization in indoor, feature-rich environments [12]. Using a PX4FLOW optical flow sensor, 64x64 pixel images were captured at 100 FPS, and an ultrasonic range sensor measured 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 algorithm developed is benchmarked against results from an open-source optical flow position estimate provided with the sensor's API [9] and are compared qualitatively using the below plot.

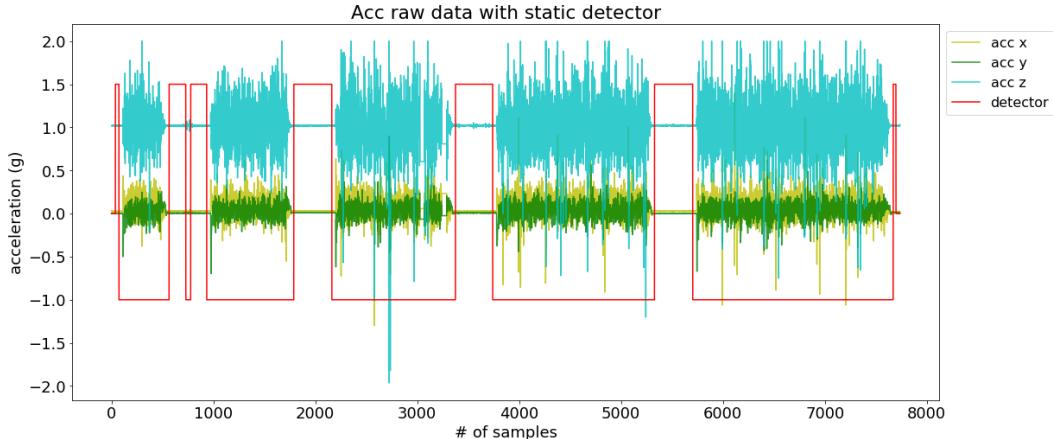


Figure 4: This should be a plot of optical flow data compared to the algorithm developed by Qingji et al.

### 3.7 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 calibrate your IMU knowing this fact. The class of filtering algorithms we explore 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

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, whereas particle filters can be more accurate if the true belief distribution is non-gaussian or if the true dynamics are nonlinear [45]. In our work, it is natural to consider the state as the position, velocity, and acceleration of the robot. We claim that this state representation satisfies the Markov condition needed by Bayesian filters, which is that our current state is sufficient to make future predictions. 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][30][32][37].

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 [28][23][43]. 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 [18]. Knowing these parameters allows one to undo distortion to the image, which is essential for detection of most AR tags.

## 4 Trade-Off Analysis

Each of the techniques presented thus far have strengths and weaknesses. In cases where those strengths and weaknesses are orthogonal, combining multiple techniques improves the overall performance. This is the fundamental principle behind sensor fusion. For example, in [21] 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 have selected for our system 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 3, 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 us, and it is unrealistic to have a consistent practice space in an FRC shop.

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 or 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 not expected to reflect.

IMUs within the budget of FRC teams suffer from accumulated drift, and as such they cannot be used in isolation. 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 complete 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, according to the specifications laid out in section 6.3, are projected to cost about \$30 (see 10). 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 moving. A procedure for this self-localization is described in section 6.5.5. 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.

Among the vision based localization systems discussed in the Literature review, 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 not perform well because the field of FRC changes over time and the robots are moving around the field during the competition. 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 align with the camera. It contains 268 pixels without payloads. 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 a storage of huge amount of data. 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. One of the most commonly used 2D barcode in robotics is fiducial tag. A system in [48] 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 almost the same as the real distance from 0.5 m to 6.5 m. The position errors were within 0.1 m from range 0 to 10 m. However, orientation errors were pretty high ( $1.5^\circ$  off) when the off - axis angle was small, but were within 1 degree from  $20^\circ$  to  $75^\circ$  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. The research, [3] 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 [50] 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. They are also generally computationally intensive. These disadvantages can be mitigated with our other localization techniques. In summary, tag based camera systems have been shown to be very accurate and we will incorporate them into our system. Published by the developers of the ArUco tag detection and pose estimation algorithm, Marker Mapper is localization technique for indoor robots. Motion capture data suggests that it is comparable to sophisticated localization algorithms such as ORB-SLAM and LSD-SLAM[31].

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

Figure 5: 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 [11]. 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 be 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 [41]. 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. Built into the sensors on mice are ways to ignore rotation since computer users want translation of the mouse to be measured over rotation in order to navigate a computer screen. Lighting is also important 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 [11].

The other option for optical flow is to use a camera which can be facing in any direction. OpenCV provides libraries and sample programs for running dense optical flow and sparse optical flow. The Lucas-Kanade method of sparse optical flow finds the best points on a frame and tracks the motion of only those points by comparing it to the motion of the pixels around that point. It is assumed all pixels around a point will have similar motion. Dense optical flow tracks the motion of all of the points on frames. Dense optical flow takes longer since it is using all of the points on a frame but can be more accurate [19]. In general, optical flow is not sufficient on its own because it does not provide global position information. However, it nicely complements our other systems because it uses a sensor we already plan to use, and provides a source of local position updates not based on any assumptions about wheels or robot dynamics.

Ultimately, we have identified IMUs, encoders, cameras with tags, beacons, and optical 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.

## 5 Experimental Results

One of the key contributions of this MQP is an extensive set of empirical and theoretical results spanning many different localization techniques. This section describes each of these experiments and explains how each test impacts the practical implementation of a complete localization system.

### 5.1 IMU Calibration

From an early experiment collecting data on a Turtlebot, we saw that double integrating the accelerometer readings would not be accurate. This was expected, because it is well known the double integration will amplify any inaccuracies in accelerometer data. Therefore, we replicated the IMU calibration procedure described in [43], which accounts for many sources of error without requiring expensive external equipment. This calibration method was relatively quick to perform, and could be performed by FRC students. This calibration

method corrects the misalignment, scaling, and biases in both accelerometer and gyroscope. This is done by solving for accelerometer calibration values that make the magnitude of acceleration during static intervals closest to 1, and then by solving 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  $T_{init}$  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 has to 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  $T_{init} = 50$ , 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  $T_{wait} = 3$  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 recorded 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 6.

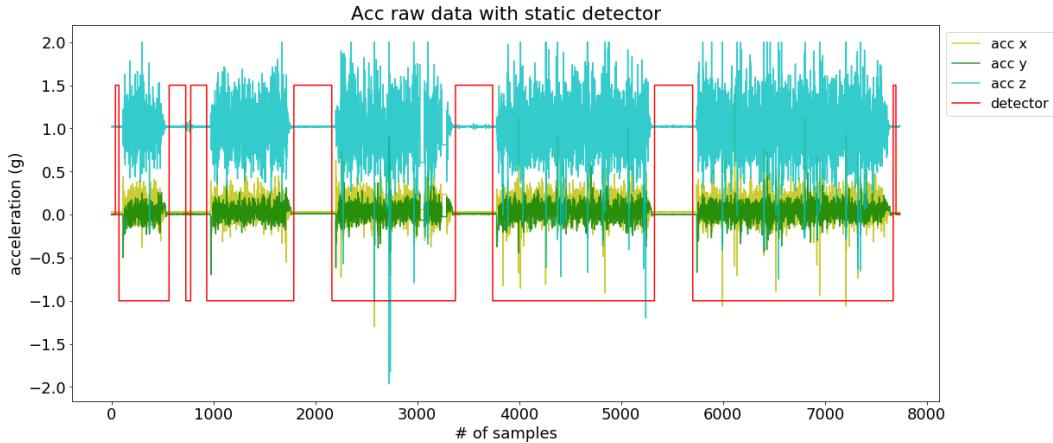


Figure 6: The black line is 1 during intervals classified as static

Using the identified static intervals, we optimize using the Levenburg-Marquedt proce-

dure 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 [43].

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

$\alpha_{yz}$	$\alpha_{zy}$	$\alpha_{zx}$	$s_x^a$	$s_x^a$	$s_z^a$	$b_x^a$	$b_y^a$	$b_z^a$
-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  $\omega_i^S$ , 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 [43] 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 7. 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 7: 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.

## 5.2 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 5.1), then a base-frame rotation. This base frame rotation accounts for the 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 gyroscope. 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 8 shows our computes yaw with the yaw reported by motion capture. Due to some strange wrap-around behavior, the mocap yaw has a small blip in value that can be safely ignored. Overall, our yaw value matches the ground truth very closely, with a maximum error of  $2.497^\circ$  in the first 1000 samples (20 seconds).

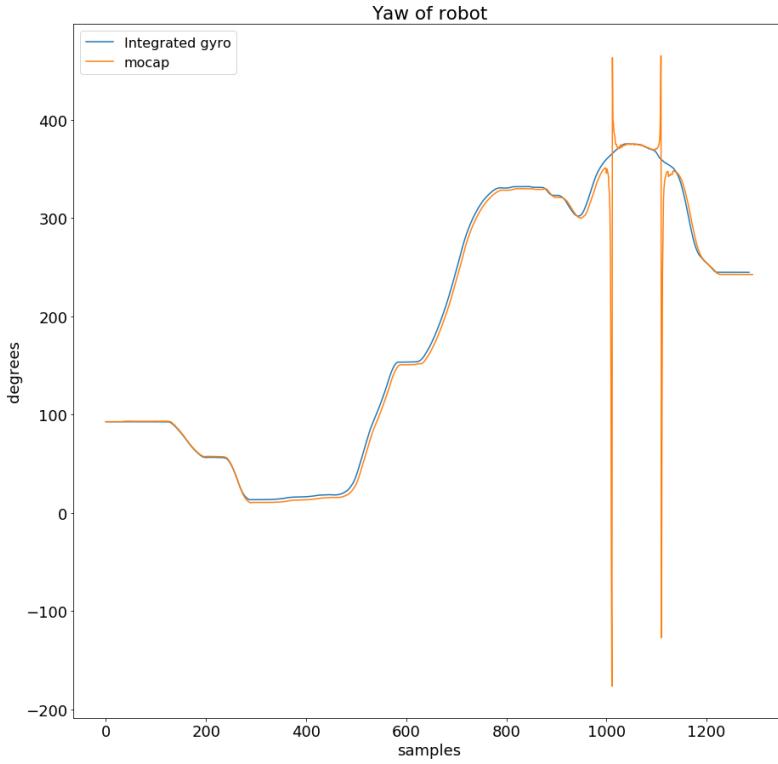


Figure 8: Comparison of yaw values between our algorithm and motion capture

Next, we compared the on-chip yaw to our integrated yaw method.

### 5.3 Characterising Drift and Bias in the Accelerometer

In this section, we demonstrate that naive double integration becomes inaccurate within  $-999\text{ s}$ , but that with calibration, base frame rotation, and zero velocity updates we can achieve accurate position information for  $-999\text{ s}$ .

#### 5.3.1 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 9). 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  $-0.000475\text{g}$ ,  $0.000158\text{g}$ ,  $0.000323\text{g}$  respectively. For reference, we note that the maximum drift of  $-0.00475\text{g}$ , 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.

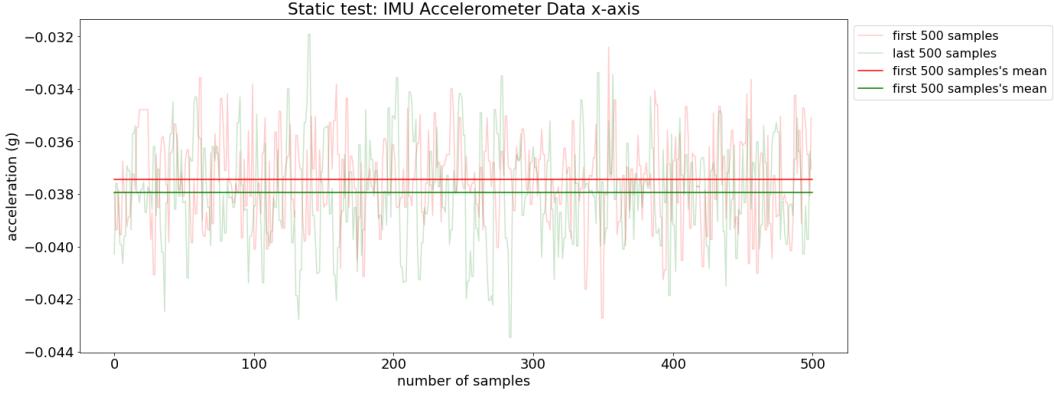


Figure 9: The raw measured X acceleration (Gs) and its mean over first and last 500 sample periods while stationary.

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 “Nypyro 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 10). 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.

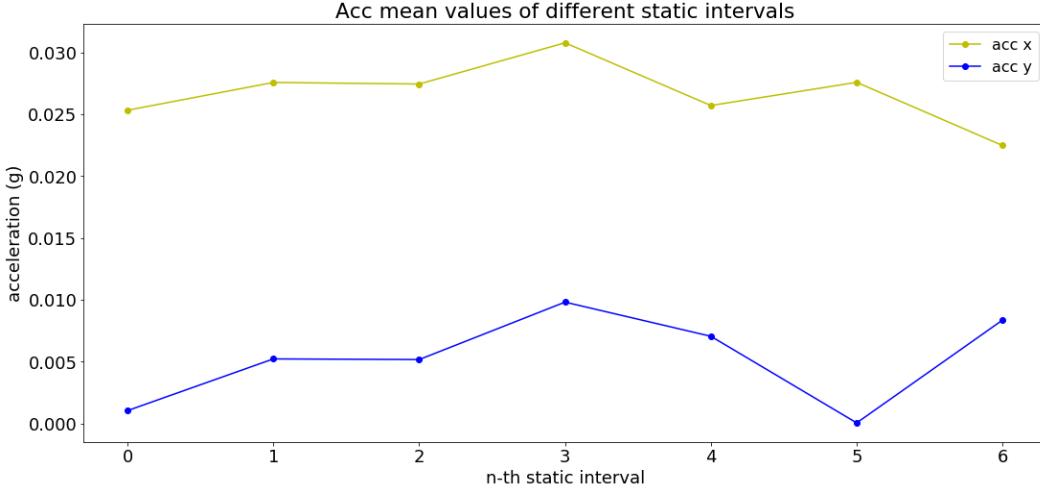


Figure 10: The means of the accelerometer data in world-frame X and Y in each static interval.

Having measured the accelerometer bias and studied its drift, we then integrated the accelerometer data with yaw angles of the “Nypyro Circles” test to see how these effect the

position. To get the best results possible, we also apply our calibration parameters (see 5.1). 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 5.2). Figure 11 and 12 show that bias and drift make velocity and displacement inaccurate after only a short period of motion.

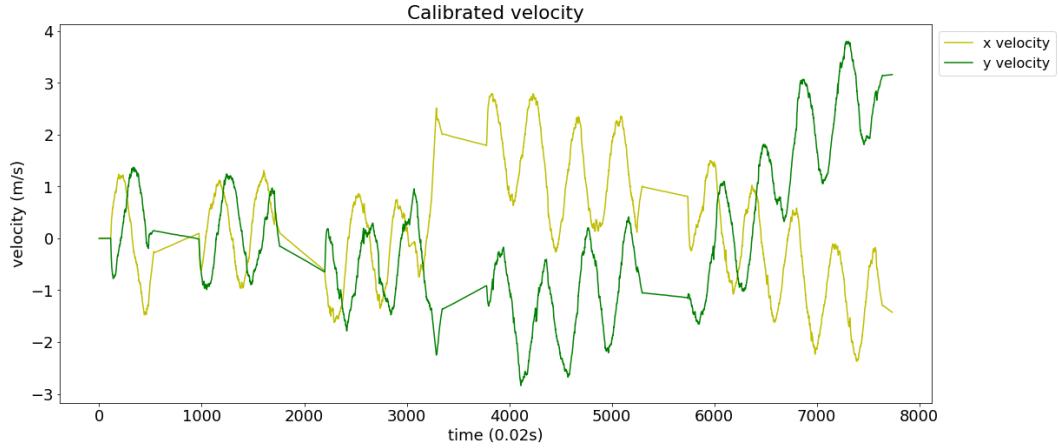


Figure 11: Velocity as derived by integrating the calibrated accelerometer measurements.

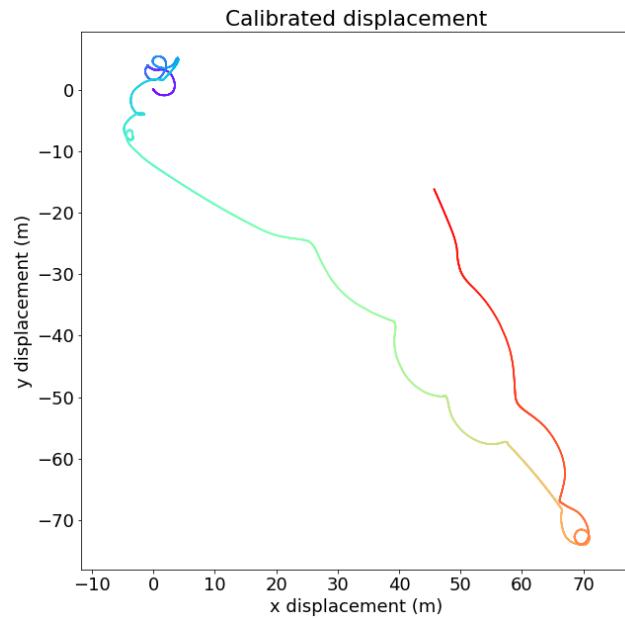


Figure 12: Displacement as derived by twice-integrating the calibrated accelerometer measurements.

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 13, 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.

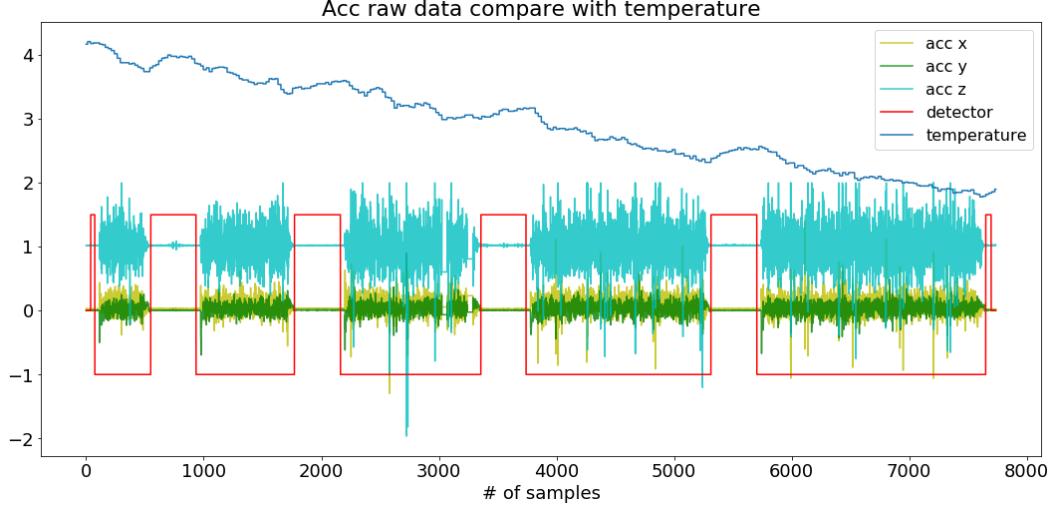


Figure 13: A Plot of temperature recorded by the NavX over the duration of our test.

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.

### 5.3.2 Zero Velocity Updates

Looking at the calibrated velocity plots (Figure 11), 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.

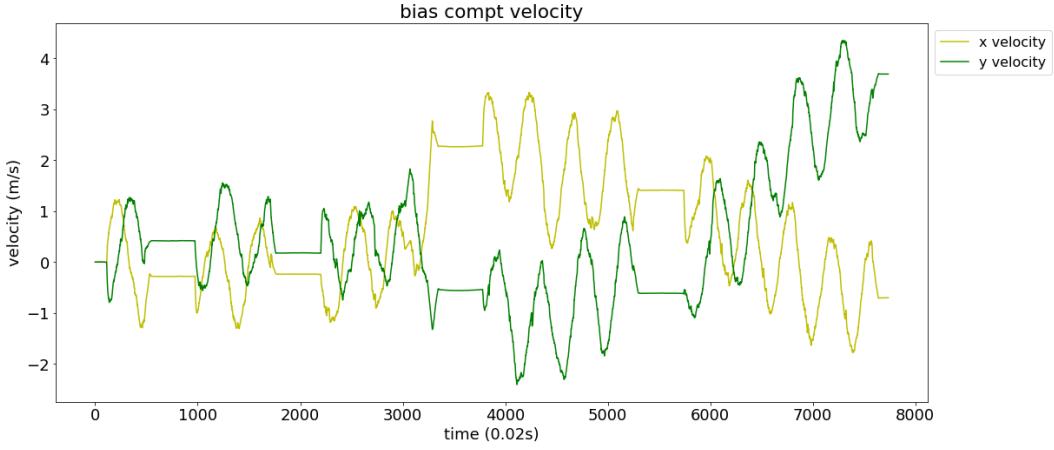


Figure 14: Velocity after bias during static intervals is removed.

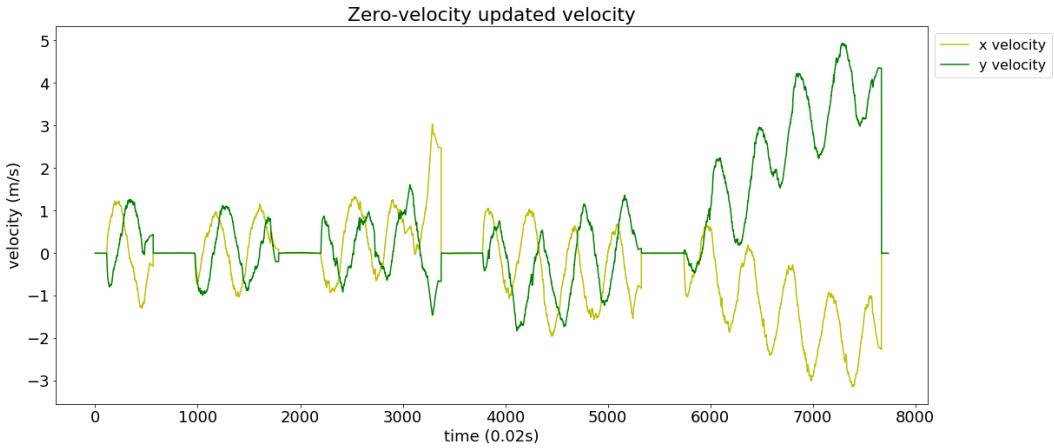


Figure 15: Velocity after both applying bias and zeroing velocity estimates.

### 5.3.3 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 5.3.1, 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) show for comparison (Figures 16, 17, 18).

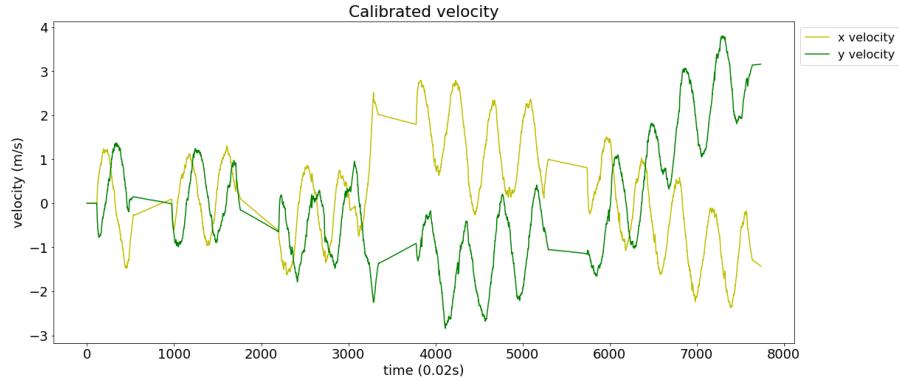


Figure 16: Velocity as derived by integrating the calibrated accelerometer measurements.

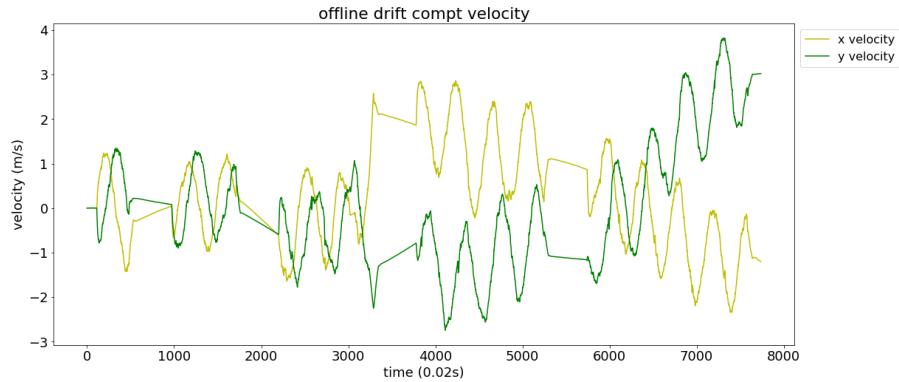


Figure 17: Velocity where drift is calculated between static intervals

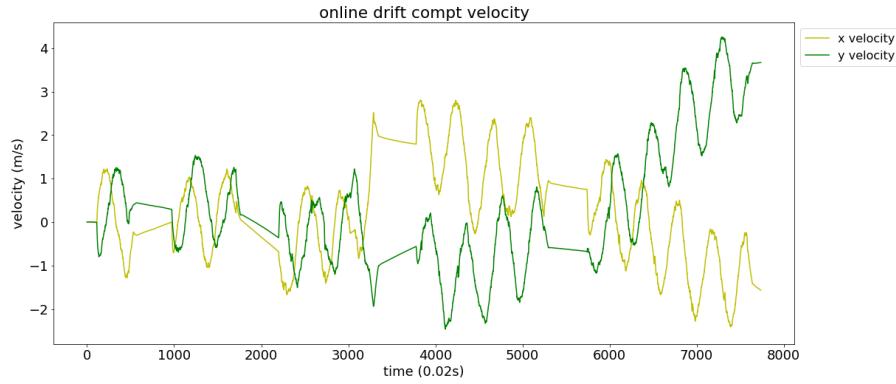


Figure 18: Velocity where drift is calculated within static intervals

#### 5.4 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 19, both methods drift significantly over the course of our experiment (“Nypro Circles”, described in Section 5.3.1). 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.

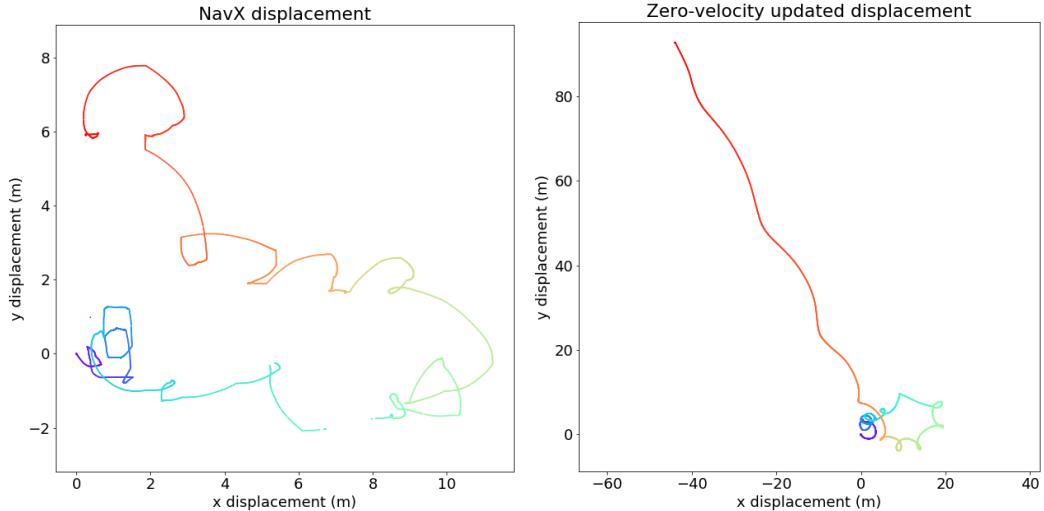


Figure 19: Comparison between NavX (left) and our method (right) over the entire experiment.

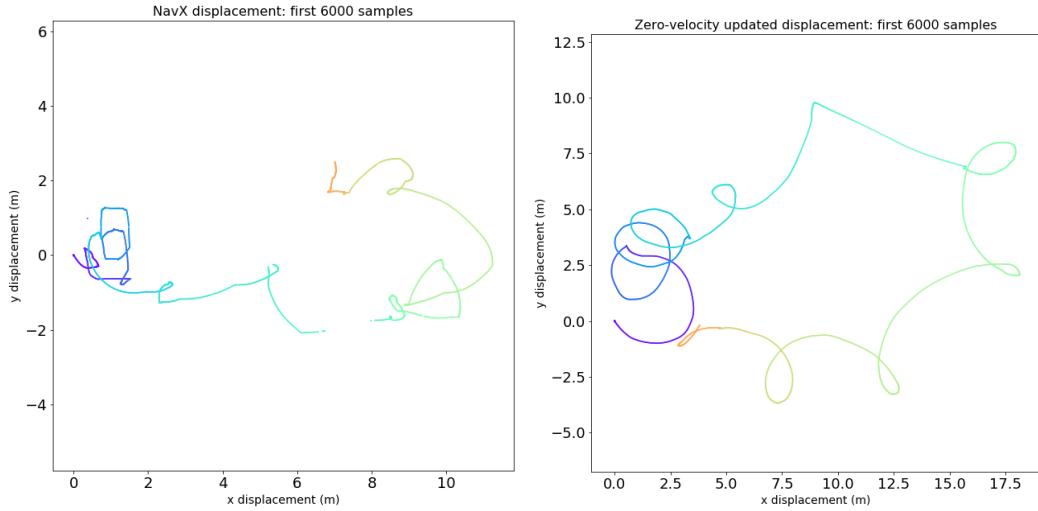


Figure 20: Comparison between NavX (left) and our method (right) over the first 30 seconds of the experiment.

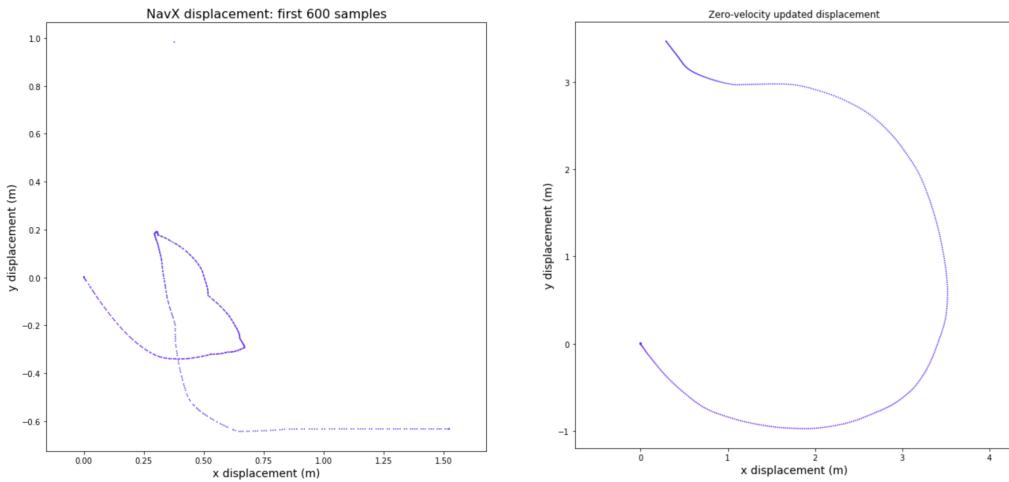


Figure 21: Comparison between NavX (left) and our method (right) over the first 3 seconds of the experiment.

## 5.5 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 42 illustrates the various delays we need to account for. We conducted experiments to estimate these delays. 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 Appendix B, 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  $\mu$ s, which we attribute to the internal circuitry of the transmitter and receiver. The variance of this delay was 16  $\mu$ s. However, we also measured delays as low as 32  $\mu$ s and as high as 79  $\mu$ 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 directly 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 24Hz 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 2.

Distance (m)	Expected Delay ( $\mu$ s)	Measured Delay ( $\mu$ s)	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 2: Measured Delays in 2kHz Sine Wave Signal

This data suggests that there is a constant delay of  $\approx 115$  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.

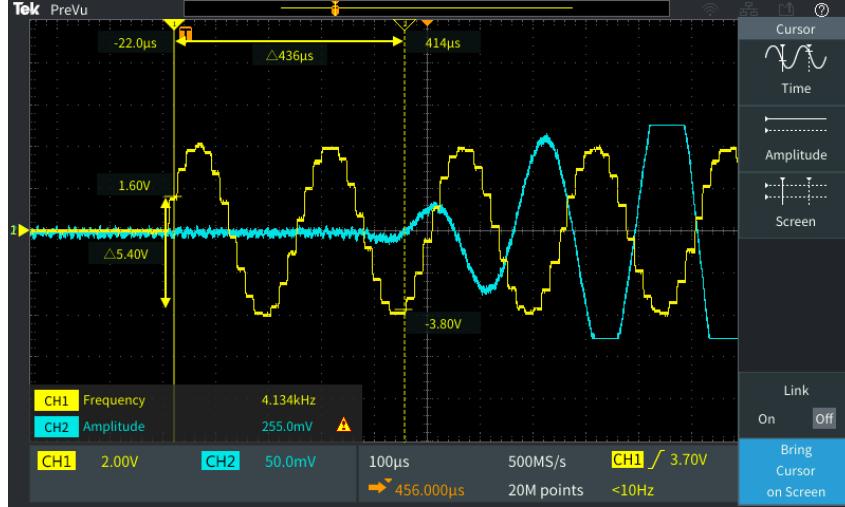


Figure 23: Capture of the measurement of ultrasonic delay on the oscilloscope

## 5.6 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 the after 27 kHz the signal is indistinguishable from the noise.

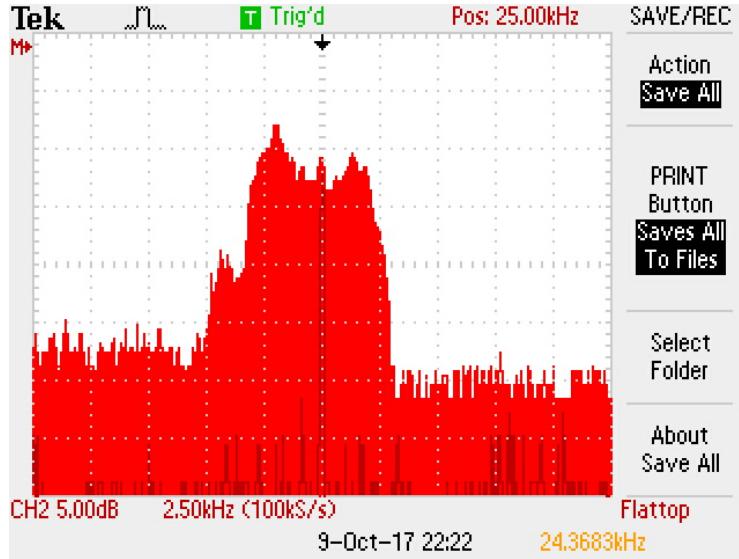


Figure 24: Frequency response of the NTX-1004PZ, centered at 25 kHz with 2.5 kHz per division. The best response is achieved at 23 kHz, and the highest detectable frequency is 27.5 kHz.

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.

## 5.7 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.

## 5.8 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 3: Time for 100 frames to run using OpenCV on laptop verse RoboRIO

We performed these measurements 5 times to check for repeatability. From these numbers, we conclude the laptop was just over twice as fast that of the RoboRIO. However, the 12 FPS rate is still fast enough for our project requirements (see Section 6.1)

## 5.9 Detecting ArUco Tags at a Distance

### 5.10 Collecting Ground-Truth with VICON Motion Capture

To evaluate the accurate 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. 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.

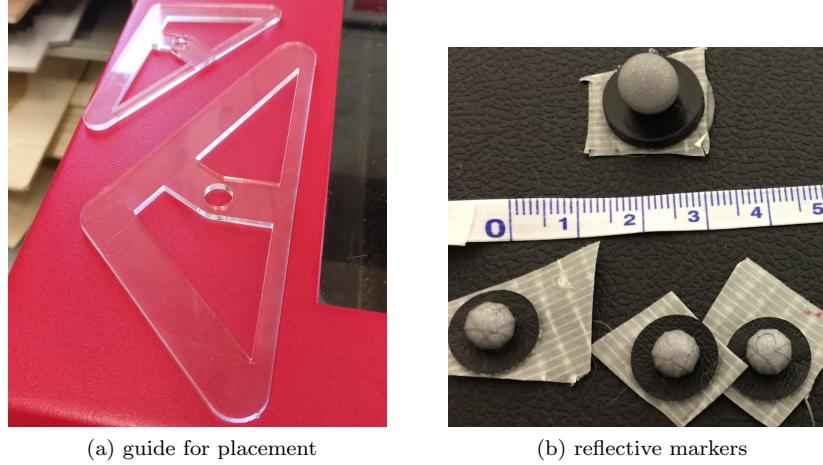


Figure 26: VICON tracker set up

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  $\approx 500 \mu\text{s}$ . Using the same markers, the pose of the ArUco tags is also measured.

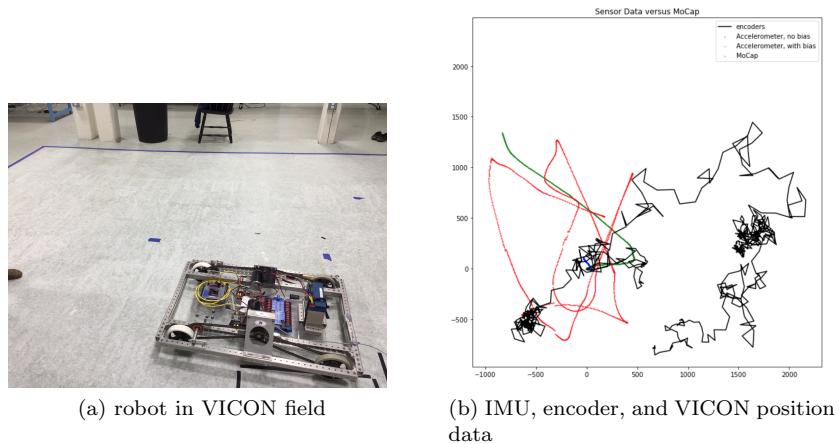


Figure 27: Collecting and Plotting Position data

## 5.11 Localizing with IMU and Encoder on the Turtlebot

## 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 5.6 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.

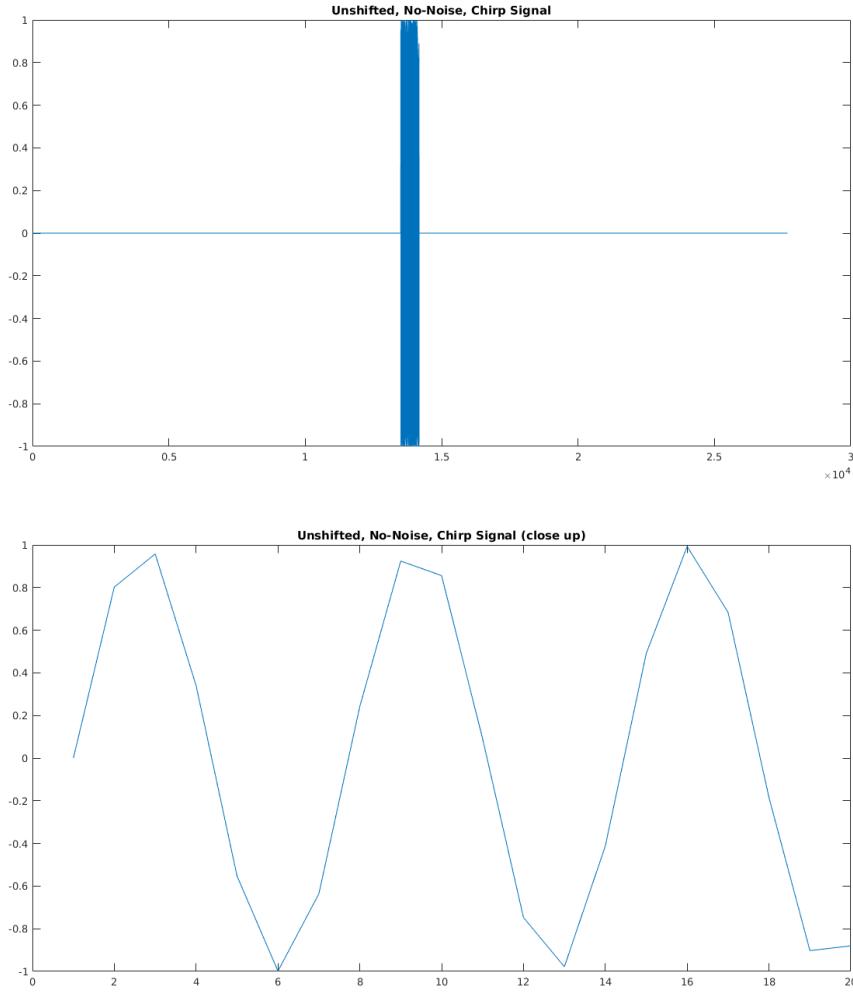


Figure 28: Unshifted, No-Noise, Chirp, 20-27kHz

Given this original signal, we then pad the signal and add noise. The result of this is

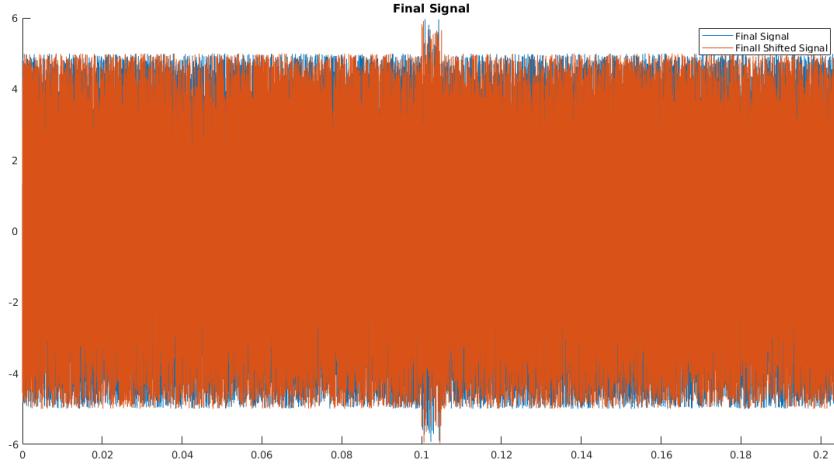


Figure 29: Both the doppler-shifted and unshifted full noisy signals.

shown in figure 29. Finally, use our original clear signal as a pattern, and convolve it with our signal. The result of this is shown in figure 30.

### 5.12.1 The Doppler Effect on Ultrasonic

Using these simulated chirps, we ask the question of whether the doppler effect of a moving FRC robot will make the signal undetectable with simple pattern matching. The plots above show what happens if a doppler shift of a robot moving  $5\text{ m s}^{-1}$  is introduced. This speed causes a doppler shift of 291.3 Hz, and after applying pattern matching using the unshifted filter we see that the chirp is detected 192.5  $\mu\text{s}$  early, which corresponds to 6.6 cm of error, which is within our requirements (see section 6.3). Furthermore, when the doppler has no effect, the error of our simulated detection is just 2.5 mm, which is well within our requirements.

### 5.12.2 Effect of Chirp Bandwidth

One of the main limiting factors of using cheap piezo speakers is the limited range of frequencies that induce a measurable response (see 5.6. Using our simulated chirps, we experimented with changing the frequency range over which the chirps sweep. When we use the full range of 20-27kHz, the amplitude of the match filter is higher and the error is lower. However, when a smaller range such as 23-24kHz is used, the amplitude of the match filter is lower, and more difficult to distinguish with a simple threshold. For the implementation of a beacon system, this means that the chirps should span as wide a frequency range as possible.

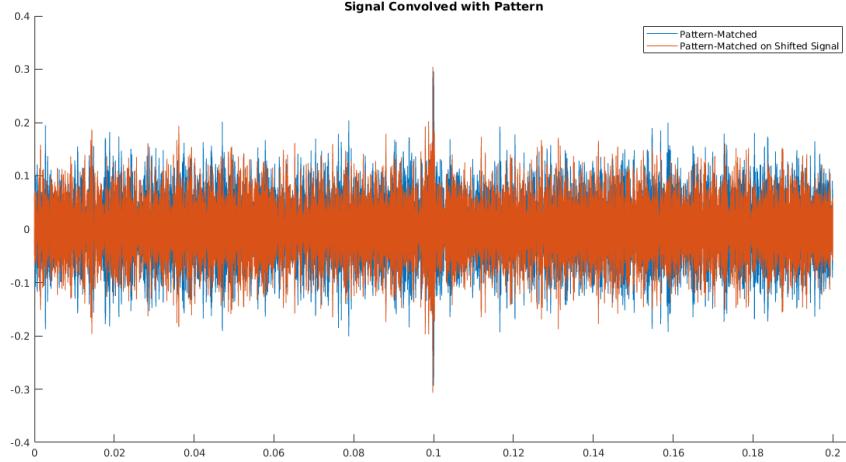


Figure 30: The peaks in the center indicate the pattern matching the noisy signals closely.

### 5.13 Detecting Ultrasonic Signals on the PSoC 5LP

### 5.14 Ultrasonic Beam Angle & Full Pattern

### 5.15 Characteristics of Piezo Transducers

Throughout this project we also discovered several interesting characteristics of our piezo speakers. First, we note that emitting square versus sine waves does not seem to effect the received signal, given the same amplitude and frequency. We tested this by connecting one piezo to a function generator and another to an oscilloscope. We generated a high frequency wave, toggling between either square or sine wave, and compared the received waveform on the oscilloscope. By simply looking at the waveform, we were unable to determine whether the function generator was in square or sine wave mode. This means that even if the transmitting speaker is being moved like a square wave, the receiving transducer will simply resonate at the same frequency and the received signal will be a sinusoidal wave. This impacts implementation because square waves can be produced with high-precision digital components rather than analog components like DACs, so one may choose to use a square wave instead of a sine wave.

### 5.16 Co-Processors for Image Processing

Since our system requires processing of images from a video stream, we evaluated the Raspberry Pi 3 and the NVidia TK1 as potential co-processors. We ruled out the RoboRIO for image processing because many teams in FRC have found the RoboRIO insufficient for vision processing, and because we did not intend for computational efficiency to be a key criteria of this project. Further still, using a coprocessor allows us to write and run whatever our system requires, irrespective of how any teams actual robot code is operating.

### 5.17 Evaluating The Placement of ArUco Tags

When doing localization with ArUco markers, generally the more markers that can be detected the better your pose estimates and maps will be. However, this is also a trade off with the amount of modification required in the environment. We would like to have as few tags as possible in our environment to minimize the amount of work required to localize in that environment. To begin to answer this question, we consider how the spacing between tags on a mock FRC field effects the detection rate of tags.



Figure 31: Tags placed on the Nypyro practice field

We placed 0.152 m tags every 1.5ft on a mock FRC field at Nypyro (see Figure 31). We recorded video driving realistically around the field and counted how frequently we detected ArUco tags. We then filtered out tags by their ID numbers to simulate spacings of 3ft, 4.5ft, and 6ft. We report detection statistics for each of these spacings based on two different runs through the field in Table 4. We also plot all the times between detections over the course of one of our runs in Figure 32. Our results show that, assuming reasonable camera settings of 480p30 (640x480, 30fps), the frequency of tag detection is essentially unchanged between 1.5ft and 6ft spacings. The only notable difference is the mean time between detections slowly rises as tags become further apart. Intuitively, this means that even 6ft between tags is close enough to expect to detect tags 10 times a second. More specifically, we can say that 95% of the time we will detect a tag every 0.1 s. We do note that during our first trial, where our camera was accidentally only recording frames at 480p8, the tag detection rate suffers more significantly as tag detection increases.

spacing (ft)	worst case (s)		95th percentile (s)		mean (s)		median (s)	
	trial 1	trial 2	trial 1	trial 2	trial 1	trial 2	trial 1	trial 2
1.5	5.100	<b>3.700</b>	0.762	<b>0.068</b>	0.235	<b>0.053</b>	0.132	<b>0.032</b>
3.0	5.231	<b>3.700</b>	0.932	<b>0.100</b>	0.269	<b>0.061</b>	0.132	<b>0.032</b>
4.5	5.900	<b>3.700</b>	1.145	<b>0.100</b>	0.284	<b>0.064</b>	0.132	<b>0.032</b>
6.0	7.832	<b>3.700</b>	1.343	<b>0.100</b>	0.335	<b>0.070</b>	0.132	<b>0.032</b>

Table 4: Tag detection metrics compared across tag spacings. The larger spacings have slightly worse performance, but still usually provide updates at least 10 timers per second. Trial 1 only recorded at 8fps, but is included for completeness. Trial 2 was 30fps.

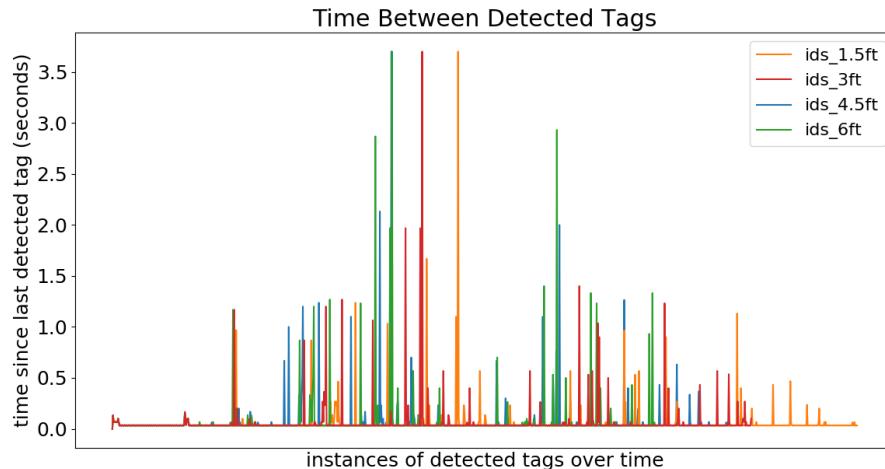


Figure 32: Times between detected tags as a function of tag spacing. Spacings between 1.5ft and 6ft perform very similarly.

It is important to note several other factors that are not explored here, including how spacing and positioning effects the accuracy of detections. Furthermore, one should ask whether the specific locations of tags, not just the spacing between them, also effects detection accuracy and frequency. Intuitively, we claim that tags should be placed in locations where the robots camera is likely to be facing, such as feeding stations and goals. However, we do not empirically evaluate this claim.

### 5.18 Statistics of CSCore Image Timestamps

We use the built-in API of CSCore to get the time stamps (in microseconds) for each image captured. During many of our tests, we logged these times to files for offline processing. We now ask how much these time stamps vary from the requested FPS. This is important information to know, because it effects whether or not one can assume a truly constant FPS. We find that there is significant variation between any individual frames. Table 5 shows key statistics about FPS over a multitude of recordings from our test robot. These

recordings are taken from our tests at Nypyro, with various requested frame rates and at various resolutions.

Requested FPS	Resolution	Mean FPS	Median FPS	Min FPS	Max FPS
30	1920x1080	14.90	14.71	9.61	22.75
30	1920x1080	15.11	14.71	10.00	27.83
30	1280x720	8.17	7.59	2.29	14.78
30	1280x720	8.35	7.60	2.00	14.75
30	800x448	29.34	31.08	4.31	31.78
60	640x480	59.43	60.02	3.53	62.48
60	640x480	59.71	60.02	3.33	89.32
30	640x480	30.00	30.01	3.76	30.13
30	320x240	30.04	31.21	2.72	31.71
30	320x240	30.03	31.22	3.33	32.30
30	320x240	30.08	31.21	14.76	32.69

Table 5: Table of statics from a multitude of CSCore streams. We find that FPS can vary throughout normal operation.

We observe that startup-lag is the true cause of low minimum FPS, and therefore does not cause significant issues unless pose estimates from the first two frames are critical. However, there are in fact cases where the camera exceeds the desired FPS but as much as 48% in the case of 60fps. There are also several cases where the processing collecting and stamping these images was not powerful enough to achieve the requested FPS. For example, we requested 720p30 on a raspberry pi, but were only able to capture at  $\approx$ 15fps. This is real constraint that must be handled in a camera based localization system, and so we report those results for completeness. However, our results show that, assuming the computer is powerful enough to achieve the requested FPS on average, there are only small variations on FPS over time. We provide two full plots of FPS over time in two of the more curious entries in table 5 to be more illustrative of how time between frames can vary (figures 34 and 33).

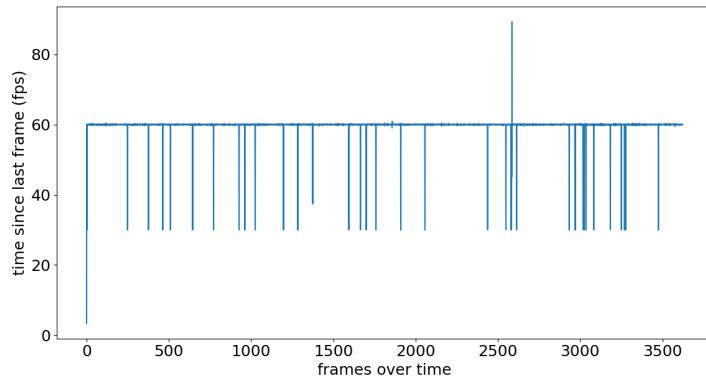


Figure 33: FPS over time for one instance of 240p30

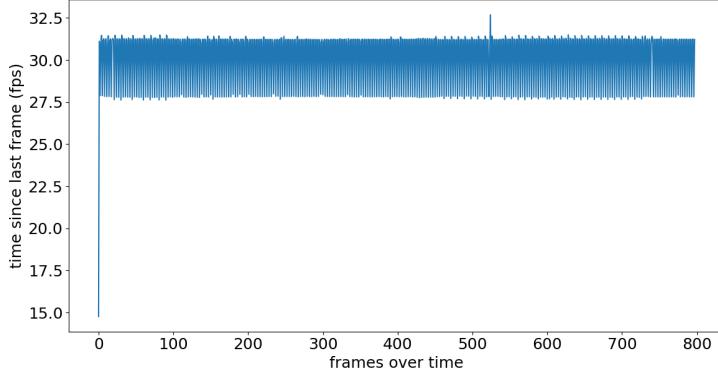


Figure 34: FPS over time for one instance of 480p60

### 5.19 Effect of Frame Rate and Resolution on ArUco Tag Detection

We explore the effect of frame rate and resolution on the frequency with which tags are detected. This frequency of detected tags is an important metric because it sets the standard for how long our dead-reckoning methods must run without receiving an update to compensate for drift. Furthermore, since the FPS and resolution of a camera correlates with cost, we want to know whether a cheap camera is sufficient for tag detection.

To answer this question, we used three cameras simultaneously recording footage from the robot as we drove around a mock FRC Field. It is important that these cameras are recording simultaneously and are mounted right next to each other, because it means the camera streams will see essentially the same view of the world, with the only variable being the resolution and FPS. The tags were placed roughly every 6 feet (according to results of section 5.17) and the robot was driven through a simulated FRC game for 60s. We then compare how frequently tags were detected by looking at gaps between detections. Table 6 shows a comparison of various key metrics between the different resolution/FPS pairings. The full plots showing all gaps in the run is shown in appendix 9.3

Condition	Worst-Case	95th percentile	Mean	Median	Mode
PS4 Eye 480p30	4.565	0.100	0.062	0.033	0.033
PS4 Eye 480p60	<b>3.049</b>	<b>0.033</b>	<b>0.032</b>	<b>0.017</b>	<b>0.017</b>
C920 1080p15	3.196	0.767	0.162	0.068	0.068

Table 6: 480p30 means 640x480 at 30fps, 480p60 means 640x480 at 60fps, 1080p15 means 1920x1080 at 15fps. The best settings by all measures was the PS3Eye camera at 60fps.

Arguably the most important metric here is the 9th percentile metric, which says that 95% of the time gaps between detected tags are less than that number. Generally, that number is quite close to the mean frame rate which means that usually you get one tag detected in every frame, but this is of course not always true. It's important to note that just because a there was a tag detected in the frame doesn't mean we get a reliable position estimate from that tag. So these numbers are not the same as the how frequently an actual position is received, which is what we truly care about.

In conclusion, the 480p60 setting performs the best by all metrics, and therefore we recommend using those settings.

## 5.20 Rate of position estimates from ArUco Tags

Although the detection of tags, is important, what is really important is the estimates of their pose. We now consider only valid position estimates, and not just any detected tag, and measure the frequency of pose estimates from our camera. We use the same recordings at the Nupro test field as we have used throughout this report, and similar to section 5.17 we compare the time between valid pose estimates. For completeness, we also compare this across the three resolution/fps settings which were recorded. One notable oddity in the data is the extremely high variance in the worst-case time between pose detections across trials and tag spacings. We simulated tag-spacings by filtering out tags based on their IDs, which is accurate because tags were placed in order. However, it means that it's possible for a tag that is present in more sparsely spaced group (ex: 6ft) to be missing from densely spaced group (4.5ft). We can explain this high variance by saying that there was a tag included in the 6ft spacing test that was not present in the 4.5ft spacing test, and without that tag there is a longer period in which no tags are seen. Secondly, we can also say that the high variance between the two trials is explained by the different paths the robot took in each trial. This is a very important result, because it means that depending on how a robot moves through the field, the frequency of valid pose estimates will change.

spacing (ft)	worst case (s)		95th percentile (s)		mean (s)		median (s)	
	trial 1	trial 2	trial 1	trial 2	trial 1	trial 2	trial 1	trial 2
1.5ft	2.7960	3.1961	0.2420	0.7736	0.1135	0.1623	0.0680	0.0680
3ft	3.5960	4.0040	0.9228	0.9680	0.1804	0.1958	0.0680	0.0680
4.5ft	6.5961	6.4640	1.0160	1.2224	0.2316	0.2556	0.0680	0.0680
6ft	10.7960	4.7259	0.8038	1.1300	0.2703	0.2625	0.0680	0.0680

Table 7: Statistics of Pose Estimates from two trials of 1080p15 footage, across various tag spacings. Note the high variance in worst-case across.

spacing (ft)	worst case (s)		95th percentile (s)		mean (s)		median (s)	
	trial 1	trial 2	trial 1	trial 2	trial 1	trial 2	trial 1	trial 2
1.5ft	1.5161	3.0489	0.0334	0.0334	0.0295	0.0324	0.0167	0.0167
3ft	1.5161	5.8312	0.0500	0.0334	0.0437	0.0448	0.0167	0.0167
4.5ft	2.9823	7.2807	0.0333	0.0333	0.0479	0.0553	0.0167	0.0167
6ft	2.2492	6.9642	0.0334	0.0334	0.0445	0.0533	0.0167	0.0167

Table 8: Statistics of Pose Estimates from two trials of 480p60 footage, across various tag spacings.

spacing (ft)	worst case (s)		95th percentile (s)		mean (s)		median (s)	
	trial 1	trial 2	trial 1	trial 2	trial 1	trial 2	trial 1	trial 2
1.5ft	3.6987	4.5651	0.0666	0.1000	0.0517	0.0621	0.0333	0.0333
3ft	4.8316	5.8313	0.0667	0.1999	0.0711	0.0899	0.0333	0.0333
4.5ft	8.1971	7.1642	0.0766	0.1500	0.1206	0.1104	0.0333	0.0333
6ft	10.4296	6.8976	0.1483	0.1517	0.1164	0.0959	0.0333	0.0333

Table 9: Statistics of Pose Estimates from two trials of 480p30 footage, across various tag spacings.

If we consider the 59th percentile metric as our most important metric, we should ask what spacing and resolution/fps settings give acceptably fast update rates. If we desire updates at least every 0.1 s (see section 6.1 for justification), then we say that 480p60 will be sufficient at any of the tested tag spacings. On the other hand, 1080p15 gives update too infrequently no matter how close tags are spaced. This makes sense, because at 15fps, a tag would need a valid pose estimate in essentially every frame to achieve 0.1 s update rate. Lastly, we can say that 480p30 probably would work with 1.5ft and 3ft spacings, and it becomes slightly too slow at 4.5ft and 6ft spacings. Ultimately, we recommend using 480p60, and suggest a 6ft spacing so as to minimize the modification of the environment.

## 5.21 Benchmarking MarkerMapper with VICON Motion Capture

## 5.22 On Building MarkerMaps

## 5.23 Erroneous detections with ArUco

Marker Maps must comprise fiducial markers from known "dictionaries" or binary encodings. Examples of several dictionaries are shown below. Dictionary selection is important because it allows users to optimize their Marker Maps. Users can choose dictionaries with different numbers of tags, marker (square) sizes, and inter-marker distances. Inter-marker distances are determined by the number of tags in the dictionary (high distances correspond to low numbers of tags). High inter-marker distances make detection more robust. Tags are defined by a list of bytes which determine the color of squares. Dictionaries can be further optimized by setting the "maxCorrectionBits" parameter experimentally to reduce false positives[34].

Throughout our experiments with ArUco tags, we accumulated several examples of detections that were erroneous in some form or another. First, we present examples of tags whose ID is misdetected (Figure 35). In all these cases the incorrectly detected ID was 2, but there is no evidence that this is an issue just with tag 2 specifically.

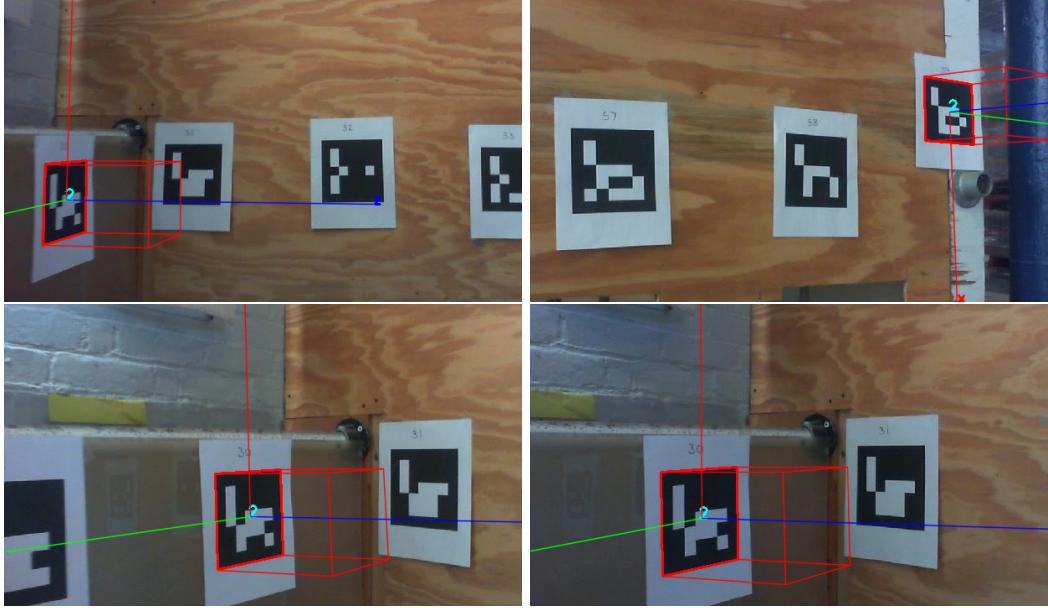


Figure 35: Tags that were detected, but with the wrong IDs

We also report how a poor camera calibration file can cause inaccuracies in the estimated poses of tags. In Figure 36, the tag's ID is identified correctly, but its orientation is incorrect.

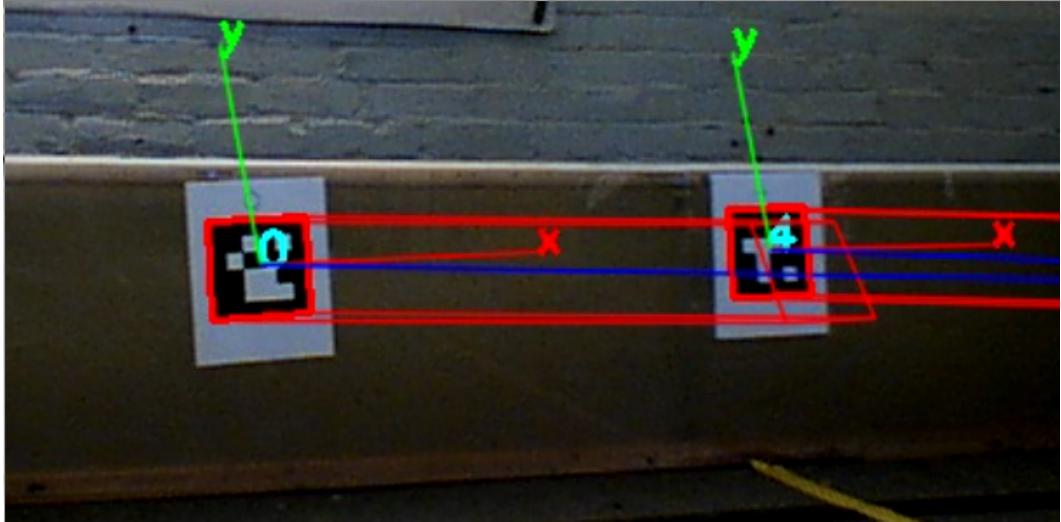


Figure 36: Example of poor camera calibration file causing skewed pose estimate.

### 5.24 Latency over the Robot Network

It is important that all of our sensor data be time stamped so we can account for latency in our state estimation. The data collected on the NavX is time stamped on the NavX, and the

encoder data is stamped when it is read on the RoboRIO. This time stamped data is sent to the TK1 over UDP. UDP was chosen because it was the easiest method with satisfactory speed. To test this, we wrote a simple program that sends 96 bytes, an upper bound on the size of all our stamped sensor data, of UDP data between the RoboRIO and the TK1. We recorded the round trip time of these packets, which can be seen in Figure 37. The round-trip latency was 0.5 ms on average, which is much faster than any of our sensors, and therefore is fast enough for us to transmit and process the data before new data arrives.

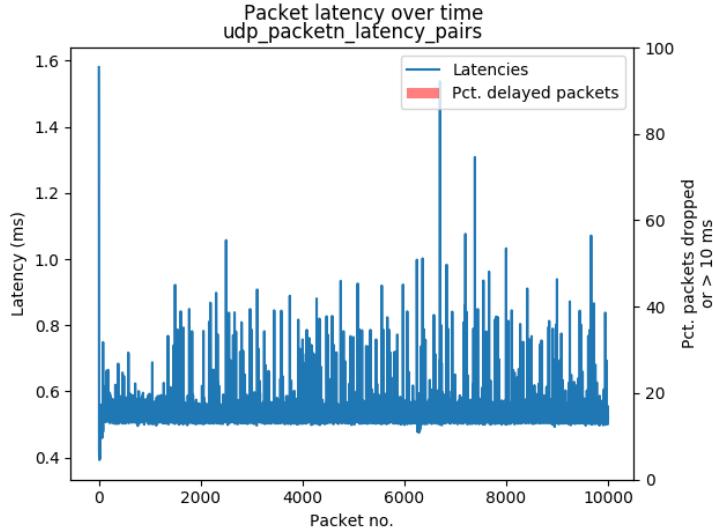


Figure 37: RTT of UDP packets between the RoboRIO and the TK1 over the robot's wired network.

Another important problem is time synchronization. The time stamps on all the data must be in reference to some common time source. To achieve this effect, we use Christian's Algorithm [6]. Specifically, we send a packet stamped with the current time on the RoboRIO to the TK1, the TK1 adds its own time stamp and responses, the and RoboRIO then add half the round trip time to the time sent by the TK1. This allows the sensor data sent from the RoboRIO to be synchronized with the clock on the TK1.

## 6 System Specification

### 6.1 Design Criteria

Here we present our goals and the criteria our system must meet in order to be successful. Broadly, we consider the following factors to be those which are important to the success of our system.

#### 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.

This goal for our system are as follows:

1. **Accuracy** of  $\pm 10$  cm and  $\pm 5^\circ$
2. **Precision** of  $\pm 5$  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. 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  $\pm 10$  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  $\pm 5$  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  $6 \text{ m s}^{-1}$ , which at an update rate of every 20 ms is a distance of  $0.02 * 6 = 0.12 \text{ m}$ . 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 most times 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 our design criteria. The full response of this survey are presented in Appendix A. In summary, the median for accuracy was 10 cm in x,y and  $5^\circ$  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 our 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  $775 \text{ ft}^2$ , 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 our system 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 the goals for our system.

### 6.1.1 ArUco SLAM

To detect the pose of a robot with respect to an absolute origin, a ArUco-based SLAM algorithm was developed. The ArUco library 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 described by the camera matrix, which is a model for a pinhole camera discussed above. The reprojection error corrects the pose estimate based on the calibrated values.

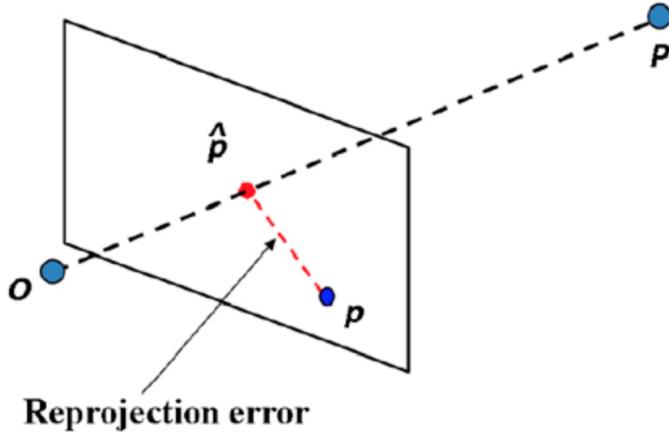


Figure 38: Calculating reprojection error [36].

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. The user must first map their work space by collecting transform data.

$$p = \begin{bmatrix} t \\ r \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \\ t_z \\ r_x \\ r_y \\ r_z \end{bmatrix} \quad (5)$$

$$T_0^1 = p_1 - p_0 \quad (6)$$

The map is complete when the user is able to query the pose from the origin using data from any tag in the workspace. Previous research indicates that calculating the transform between tags is sufficiently accurate. This protocol is preferred because it does not require the user to manually measure distances between tags and input them into the program. Instead, the user generates the map using the camera. Presently, the algorithm fails to utilize redundant information and instead computes position based on the first tag detected in the image. Future progress includes utilizing a least-squares solver for optimizing the pose estimate using information from all tags in a frame. Finally, to obtain the pose of the robot from the camera, the transform between the camera and the centroid of the robot must be computed.

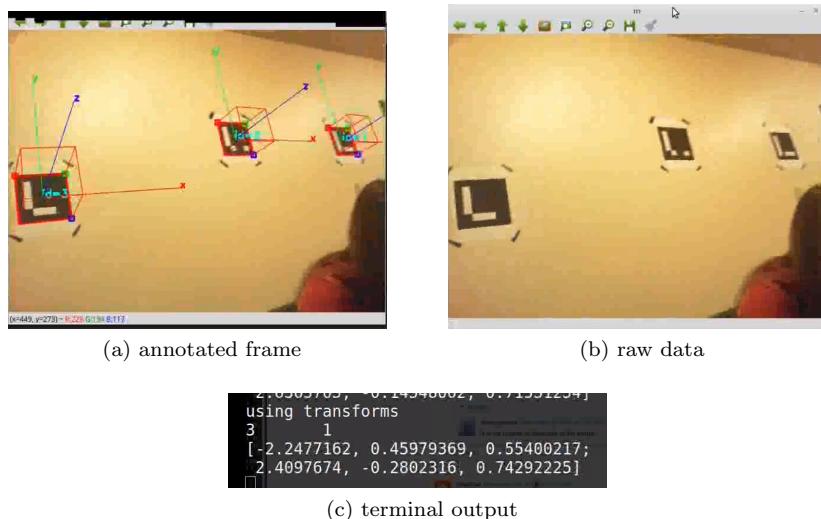


Figure 39: Visualization of pose estimate

## 6.2 Kalman Filtering

### 6.3 System Requirements

Based on the extensive literature review, the few preliminary experiments we've conducted, and our design criteria, we eliminated our initial list of possible techniques down to the following five:

- ## 1. Radio and ultrasonic beacons

2. IMU
3. Drive wheel encoders
4. Optical flow
5. Camera with matrix tags

We have found examples of each of these techniques being used successfully, and in many cases have verified that they satisfy our criteria for accuracy, precision, update rate, and accessibility criteria. We are confident that any combination of these methods would work. These techniques are complementary in their sources of error, and together we believe they will make a robust localization system.

## 6.4 Timeline and Goals

To maximize the likelihood that our project is complete in timely manner and with rigorous methodology, we have drafted a set of goals and a timeline to help us manage our time. Below we present our set of *must* goals, which we have agreed are all required for our project to be a success (in no particular order).

- Support 4 wheel differential drive robots moving at continuous driving speeds of up to  $3 \text{ m s}^{-1}$ .
- Support the NavX.
- Use Kalman filtering to combine each sensors estimate of position into one estimate.
- Differentiate between tags with the camera and measure pose relative to them with a standard 720p webcam.
- Beacons provide global position updates.
- Use the out of the box OpenCV API to get positions of keypoints in the frames.
- Provide a GUI for specifying the configuration of the sensors on the robot and the configuration of the field or practice space.
- Accurately describe the conditions under which camera localization will work, especially with respect to jitter and occlusion.
- Support getting one fused position update
- Specify what kinds of sensor mounting configurations are supported
- Provide the designs and the code for the beacons.
- Provide the optimal analysis or coverage of an empty FRC field and the 2018 field
- Fail gracefully when some of the sensing techniques are unreliable or failing

## 6.5 Implementation Details

Having decided on the sensing techniques we will use, we now describe our detailed system Specification. Ideally, the description in this section serves as the full plan for our implementation of the system during B Term, and most of the major design decisions have been made and clearly presented.

### 6.5.1 The Test Robot

This section describes the hardware and software of the test robots we used throughout our experiments. We use an old FRC Kit-of-parts Chassis with 6” wheels. The rear wheels have high-grip rubber and the front wheels are a slick hard plastic. This creates an interesting uneven turning center, which is a good test-platform for robots that slip when turning. We use two CIM motors to drive the wheels and two Talon speed controllers to drive the CIMs. Our robot also has a RoboRIO and an NVidia brand Jetson TK1, which are networked together with a COTS router. For sensing, we have two Greyhill 63R256 encoders, a cheap 720p USB webcam, and a NavX-MXP. Figure 40 shows the fully assembled robot.

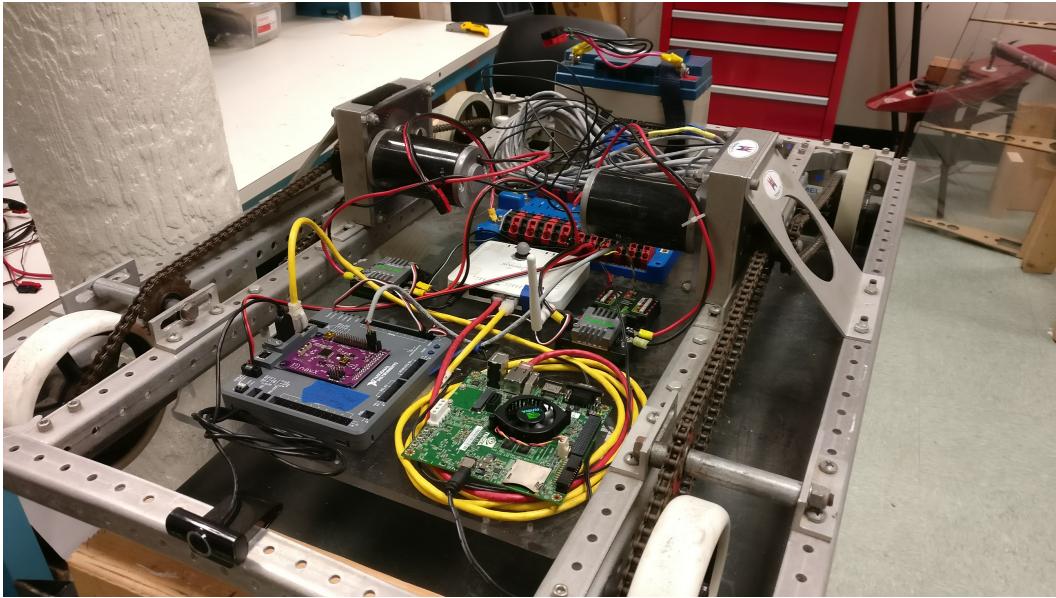


Figure 40: Test Robot used in our experiments

### 6.5.2 General Architecture

Given the five sensing technologies mentioned above, we will now provide details on the configuration and use of each sensor. The Navx IMU is connected to the RoboRIO, which is the main processor which are FRC teams are required to use. The encoders on the drive train are similarly connected to the digital IO pins on the RoboRIO. These sensors are connected to the RIO directly because they are often used by FRC Teams in their robot programs. We also introduce the NVIDIA Jetson TK1 as a co-processor. The camera is connected via USB-A to the TK1. Because of its superior computational power, the TK1

is responsible for processing all of the raw sensor data and delivering the result back to the main robot program. In this setup, the TK1 can receive sensor data from the RoboRIO over the robot network band from the USB webcam directly.

There are two software components of our system. First, there is a a C++ library cross-compile for the RoboRIO that FRC teams will use in their robot projects. Teams are expected to call two or three simple functions in their robot program, which gives us everything we need to log sensor data and send it to the TK1. Because of this minimal API, we require very few changes to robot programs in order to get localization. This will make it approachable for teams and encourage them to try localization. The second software component is a C++ program running on the TK1. This program reads the camera data, serves http camera streams, receives the sensor data from the RoboRIO, computes the position of the robot, and reports this position over network tables. Again, the TK1 was chosen for this task because of it's computing power.

### 6.5.3 NavX Processing

In this section we describe our final methods for incorporating the accelerometer readings and fused yaw readings from the NavX IMU. In short, the final process is to apply calibration parameters, rotate the accelerometer readings into the robot base frame, apply drift and bias compensation, the feed these readings into our extended kalman filter. Experimental results on each of these steps are presented above in Sections 5.1, 5.3.1, and 5.3.2, but here we describe the final process in greater detail.

The first step is to apply the calibration values, which is a straightforward application of Equation 7. These parameters are determined a priori by the process described in Section 5.1.  $a^S$  is a 3 by 1 vector of the raw accelerometer readings, and  $a^O$  is the same vector now calibrated. This calibration process accounts for axis misalignment, scaling, and bias errors.

$$a^O = T^a K^a (a^S + b^a) = \begin{bmatrix} 1 & -\alpha_{yz} & \alpha_{zy} \\ 0 & 1 & -\alpha_{zx} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x^a & 0 & 0 \\ 0 & s_y^a & 0 \\ 0 & 0 & s_z^a \end{bmatrix} \left( \begin{bmatrix} a_x^S \\ a_y^S \\ a_z^S \end{bmatrix} + \begin{bmatrix} b_x^S \\ b_y^S \\ b_z^S \end{bmatrix} \right) \quad (7)$$

After calibration, we account for the NavX not being mounted in the same orientation as the robot's base frame. We would like the positive accelerometer readings in X to point "forward" in the robot frame, and the positive readings in Z to point upwards. To achieve this, we assume the robot is stationary for the first 5 s of being turned on, and record the accelerometer readings during this period. We then use the Equation 8 to find the rotation that rotates those measured accelerations into the expected acceleration in the robot base frame. This expected acceleration in the robot frame is  $[0, 0, 1]$ . We cannot infer the yaw of the robot from this process, so we assume the yaw offset between the NavX and the robot is given to use a priori.

$$R = I + [v]_{\times} + [v]_{\times}^2 \frac{1}{1+c} \quad (8)$$

$$[v]_{\times} = \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix}$$

$$v = a \times b$$

$$c = a \cdot b$$

After calibration and base frame rotation, a static detector was applied to all of the calibrated and base-frame-rotated data to classify the static and dynamic intervals. In this static detector, we used  $\varsigma^6$  as the threshold and 80 samples as the sliding window, where  $\varsigma$  is the magnitude of the variance of an example static period (Equation 9). We chose  $\varsigma^6$  experimentally by raising the power, which lowers the threshold, until the intervals were sufficiently static.

$$\varsigma = \sqrt{[\text{var}(a_x^t)]^2 + [\text{var}(a_y^t)]^2 + [\text{var}(a_z^t)]^2} \quad (9)$$

We next applied zero velocity update on the data. Zero velocity update removed bias on every data sample by assuming that the bias of a data sample is the same as the mean bias of the most recent static interval. Zero velocity updates also involve resetting the velocity estimate to 0, which could be implemented as a hard-reset or a probabilistic EKF measurement update with near zero variance to indicate high confidence.

Lastly, we applied clockwise rotation matrix  $R(\theta)$  on every accelerometer data to transfer the direction of accelerometer data into the world frame, where  $\theta$  is the yaw reported by the NavX function `GetYaw()`.

#### 6.5.4 Encoder IMU Fusion

The fusion of Encoder and IMU data will be the foundation of our localization system. While we do not plan to explore it too deeply, it will be the first component we work on in B term. The sensors supported will be the Kauai Labs NavX MXP and Micro. The system will not prevent the user from interfacing with either sensor in other ways. The IMU and encoder toolkit will feature an interface for user to input information about their kinematic base, namely wheel radius, wheelbase length, and encoder resolution. Using this information, the system will generate a state space model comprising minimally position and velocity terms. The drivetrain configuration that will be supported is a differential four wheel drive. Support may later be offered for one omnidirectional configuration, such as H-Drive. Using this information, a Kalman and complementary filtering program will be developed. Inputting a sensor configuration in software will allow a team to display local trajectories using a 2-D visualization. The team must be able to quickly modify parameters such as the filtering function used or the radius of a wheel and recalibrate each sensor online. This portion of the project must be completed approximately at the start of the FIRST build season.

#### Calibration

#### Description of the Robot Model

### 6.5.5 Radio and Ultrasonic Beacons

Important implementation details of the beacon system include the number of beacons needed, the radio communications protocol, and the ultrasonic signal format. The number of beacons can either be picked to minimize cost, and then the beacons can be designed to meet this criteria, or the characteristics of the beacons can be determined and the number of beacons needed derived from that. In order to support arbitrary practice areas, the second method is preferred. In this section we cover relevant calculations and possible implementation details for beacons.

#### Overview of components

Each beacon consists of a PSoc5 LP processor, NTX-1004PZ piezo transducer, XY-MK 433 MHz radio transmitter, XY-FST 433 MHz radio receiver, and other supporting components. During normal operation, each beacon will listen for a radio signal telling it to emit an ultrasonic signal. We expect that a few other components such as resistors, LEDs, and a battery will be required for the final beacon.

#### Format for Ultrasonic Signal

The ultrasonic signal should be designed to maximize both the distance it can be detected from and accuracy with which it's timing can be detected. Using a simple pulse of fixed frequency and amplitude is not ideal because it trades off energy transmitted (and thus distance) with accuracy. This is because a longer pulse contained more energy and can be detected from further, but at the same time the receiver cannot distinguish timing within the length of the pulse. Therefore, a chirp signal will be used. Compare the usual function relating amplitude to time of a sine wave to the quadratic chirp signal.

$$x(t) = A \cos(\omega t + \phi)$$

$$x(t) = A \cos\left(2\pi\left(\frac{f_1-f_0}{2T}t^2 + f_0t\right) + \phi\right)$$

This function will generate a chirp with amplitude  $A$ , starting from frequency  $f_0$  going to  $f_1$  over time interval  $T$ . Figure 41 shows the specific waveform with the parameters we expect to use. This is a linear chirp signal because the instantaneous frequency of the signal changes linearly with time.

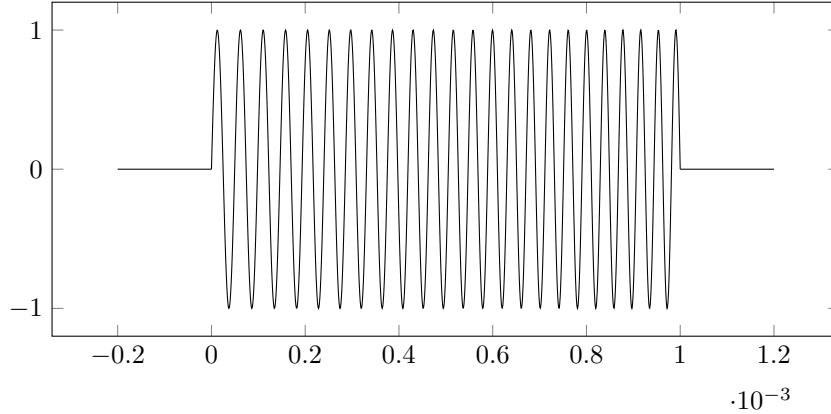


Figure 41: Chirp signal with  $\phi = 0$ ,  $A = 1$ ,  $f_0 = 20$  kHz,  $f_1 = 27$  kHz, and  $T = 1$  ms

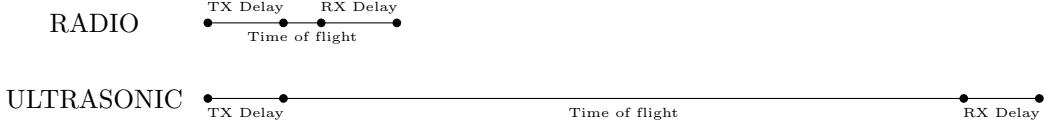


Figure 42: Timing of radio and ultrasonic signals. Experiments indicate 46.175  $\mu\text{s}$  total RF delay and 1 ms total ultrasonic delay.

Item	Cost per Beacon
PSoc 5LP	\$10.00
RF Tx/Rx Pair	\$1.68
piezo speaker	\$1.65
9v battery	\$1.19
battery connector	\$0.54
LCD display	\$3.90 (optional)
resistors and capacitors	\$5.00 (estimate)
prototyping board	\$5.00 (estimate)
Total	28.96

Table 10: estimated bill of material for beacons

This is the waveform we will be emitting from the ultrasonic sensors. The benefit is that it can be high power and still allow the receiver to determine precisely where in the waveform it is listening. One method for doing this is to take the discrete short time Fourier transform, which will show us how the power at various frequencies changes over time. By applying the FFT at a particular instant during the chirp, we can determine the position in time within the chirp of the current FFT. For instance, in the signal above if we apply and FFT at time  $t_n = 1.0$  and discover high power at frequency  $f = 22 \text{ kHz}$ , we know that the chirp began at time  $t_0 = t_n - \frac{f-f_0}{f_1-f_0}T = 1.0 - \frac{22-20}{27-20}0.001 = 0.99857$ . Another simpler strategy is to slide the waveform you expect to hear across samples of recorded raw input and check for the point of highest correlation. In other words, slide the waveform in Figure 41 and match it to the received signal. Both of these techniques are used in practice, and we are considering both in our implementation.

Our beacon system relies on accurately knowing the distance to a beacon, and therefore knowing exactly when the start of the signal left the transmitter and when the start of the signal arrived at the receiver. Figure 42 shows the sources of delay we are accounting for, and we describe our methodology for measuring these delays in the Experimental Results section.

### Path Loss

We calculate the free space path loss (FSPL) of the radio signals at the farthest distance from the beacons. For this calculation, we assume the worst case where the beacon is on the other side of the length field, which is 16.5 m away. Over the more realistic distance of half the width of the field (5.6 m), the path loss is only 40 dB.

$$\text{FSPL} = 20 \log_{10} \left( \frac{4\pi R f^2}{c^2} \right) = 20 \log_{10} \left( \frac{4\pi * 16.5 * 433 \times 10^6 \text{ Hz}}{3 \times 10^8 \text{ m s}^{-1}} \right) = 49.521$$

### Self-Localization of Beacons

Beacons that can automatically discover each other and their relative positions will tremendously improve the ease of setup for the beacon system. We believe this is possible, and in this section we describe a protocol for doing so.

When the first beacon is turned on, it will listen for radio packets for a brief period of time to check if it is the first beacon to be turned on. If it hears nothing, it will assume it is the first beacon and assign itself ID 0. This beacon will be responsible for sending the synchronizing radio pulse as well as handle the distribution of IDs to the other beacons. At this point, beacon 0 will send packets advertising that ID 1 is available. When the next beacon is turned on, it will get the message that ID 1 is available, assign itself that ID, and respond with an ACK message confirming that ID 1 is now taken. Beacon 0 will then begin advertising that ID 2 is available. This process continues for N beacons. Once each beacon has its assigned ID number, beacon 0 will start organizing the self-localization of each beacon. First, beacon 0 will send a packet telling everyone that beacon 0 will transmit ultrasonic. All beacons other than beacon 0 will hear this and start a timer, and beacon 0 will itself transmit ultrasonic. Each beacon will receive this ultrasonic and compute its distance to beacon 0. After waiting a fixed period of time (such as 50 ms) for this process to complete, beacon 0 will send a packet telling everyone that beacon 1 will transmit ultrasonic. This process continues until every beacon has recorded its distance to every other beacon. At this point, beacon 0 will tell each beacon sequentially to report its distance calculations to beacon 0. Beacon 0 will then compute the location of every beacons by minimizing the least squares error in the system of equations defining the beacon locations. Finally, it will report the resulting positions to all the beacons. These communications can also be monitors from a laptop with an inexpensive USB software defined radio, such that the status and result of this process can be shown in a GUI and debugging information can be communicated.

#### **Number of Beacons Needed**

The number of beacons can be decided once the range and beam pattern of the beacons is known. To do this, a rigorous test that measures the sound level at various angles and distance will be performed. Once this is known, we can offer a tool or set of steps to determine how many beacons are needed and where beacons should be placed.

#### **6.5.6 User Interface (GUI)**

As mentioned above, one of our goals is to develop a GUI. This GUI will serve several functions, such as displaying the position of the robot on the field, showing debug information from each subsystem, and allowing students to input parameters about their robot and their practice space. For example, doing forward kinematics with the encoders requires that the position of each wheel and encoder is known. Additionally, the GUI could allow a visual interface for the students to say where and how their practice space maps to the imaginary FRC field.

## **7 Conclusion**

Over the course of A Term, we conducted thorough background research on various methods for localization and evaluated their performance. Based on numerous reports for each localization technique, a survey of FRC students and alumni, and our own knowledge of the challenges of localization for FRC we narrowed down the list of possible techniques to encoders, IMUs, beacons, cameras and tags, and optical flow. Each of these methods has been shown to work in other indoor mobile robot environments, and together they make

up for the weakness of each individual method. Our strategy will be to apply each of these five techniques to the degree we believe is necessary to meet our design criteria, rather than to choose one method and optimize it deeply. Our performance will be judged against the design criteria stated above, and we have developed goals and a timeline to guide our work moving forward.

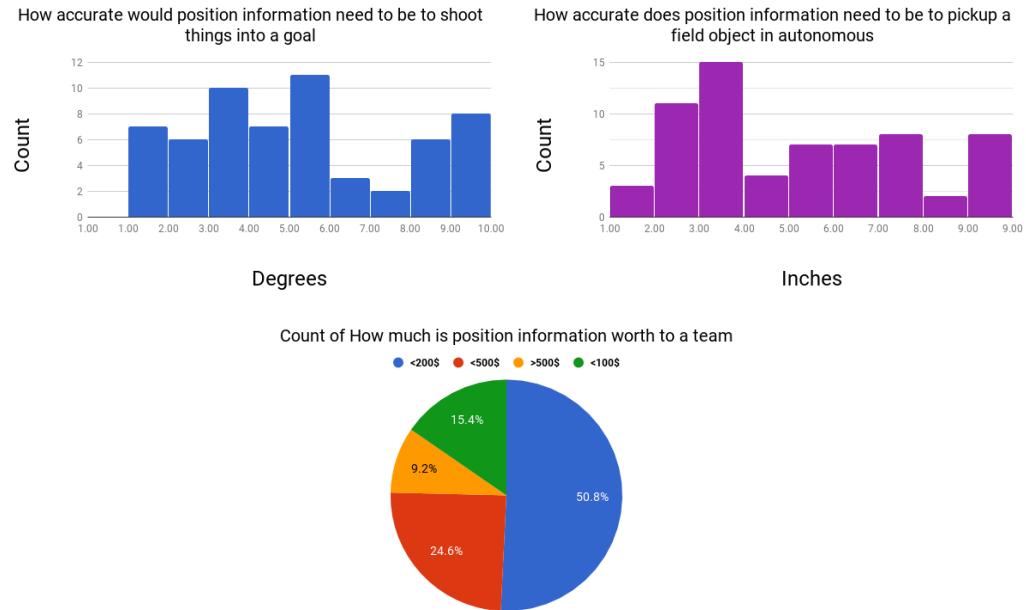
## 8 Acknowledgements

We thank our advisors, Bradley Miller and William Michalson for their guidance. We also thank our sponsors, AndyMark and Kauai Labs for their generous donation of hardware. We'd like to thank Scott Libert and Eric Peters for their advice and guidance. Finally, we give a special thank you to FRC Team 261 Gael Force, for letting us use their practice field.

## 9 Appendix

### 9.1 Appendix A

#### Survey Responses

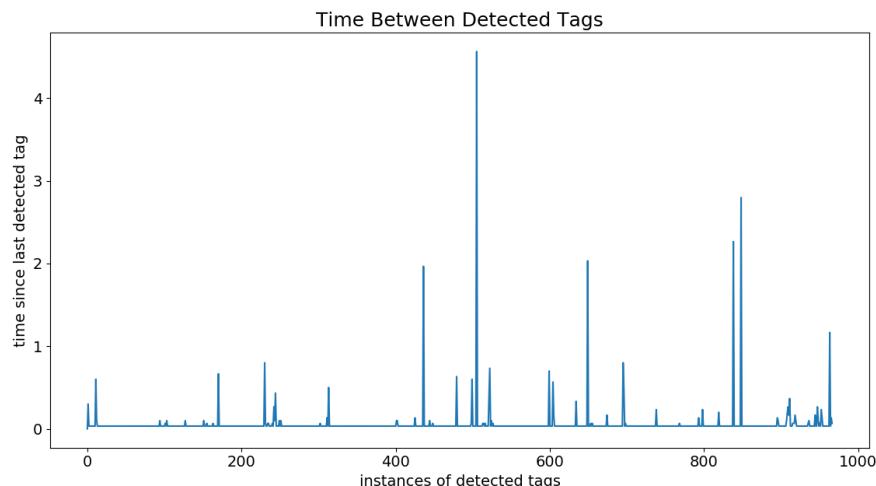


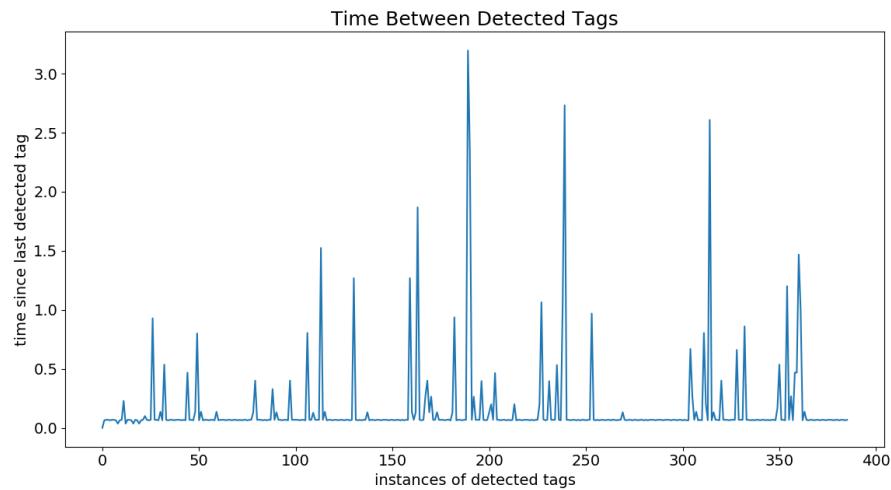
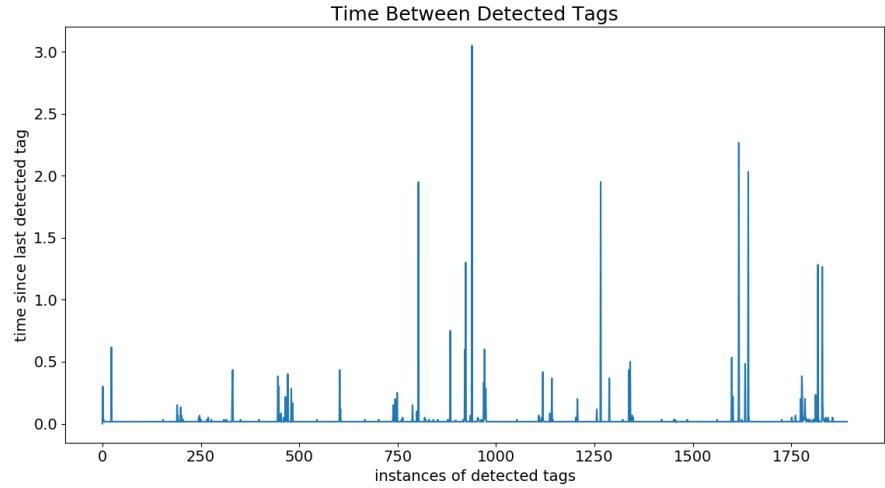
## 9.2 Appendix B

Measured Distance (m)	Measured Total Time ( $\mu$ s)			Average Delay
0.0630	45.44	42.80	34.48	40.90646
0.1425	52.72	50.48	52.09	51.76286
0.1505	64.16	63.36	60.24	62.58616
0.2210	40.33	36.79	36.40	37.83926
0.2415	49.52	45.76	43.92	46.39919
0.2460	47.47	53.84	44.71	48.67251
0.2965	34.36	34.00	43.76	37.37234
0.3085	79.36	62.16	59.52	67.01230
0.3390	39.92	57.27	38.96	45.38220
0.3770	41.75	40.75	45.53	42.67541
0.3600	38.40	38.40	37.68	38.15880
0.0070	35.60	36.08	34.32	35.33331
Average Delay ( $\mu$ s)				46.175

Table 11: The time of flight of radio over tens of centimeters is insignificant compared to the delay within the transmitter and receiver.

## 9.3 Appendix C





## References

- [1] P. Bahl and V. N. Padmanabhan. RADAR: an in-building RF-based user location and tracking system. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, volume 2, pages 775–784 vol.2, 2000.
- [2] Billur Barshan and H. F. Durrant-Whyte. Inertial Navigation Systems for Mobile Robots. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, 11(3):329–350, September 2017.

- [3] Changsheng Chen, Alex C. Kot, and Huijuan Yang. A two-stage quality measure for mobile phone captured 2d barcode images. *Pattern Recognition*, 46(9):2588–2598, September 2013.
- [4] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. IMU Preintegration on Manifold for Efficient Visual-Inertial Maximum-a-Posteriori Estimation. *Robotics: Science and Systems*, 2015.
- [5] C. K. Chui and G. Chen. *Kalman filtering: with real-time applications*. Number 17 in Springer series in information sciences. Springer-Verlag, Berlin ; New York, 2nd ed edition, 1991.
- [6] Flaviu Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3(3):146–158, September 1989.
- [7] Duarte Dias and Rodrigo Ventura. Barcode-based Localization of Low Capability Mobile Robots in Structured Environments. *2012 International Conference on Intelligent Robots and Systems*, 2012.
- [8] E. DiGiampaolo and F. Martinelli. Mobile Robot Localization Using the Phase of Passive UHF RFID Signals. *IEEE Transactions on Industrial Electronics*, 61(1):365–376, January 2014.
- [9] Dominik Honegger, Lorenz Meier, Petri Tanskanen, and Marc Pollefeyns. An open source and open hardware embedded metric optical flow CMOS camera for indoor and outdoor applications. *2013 IEEE International Conference on Robotics and Automation*, May 2013.
- [10] M. Drumheller. Mobile Robot Localization Using Sonar. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(2):325–332, March 1987.
- [11] Davinia Font, Marcel Tresanchez, Tomàs Pallejà, Mercè Teixidó, and Jordi Palacín. Characterization of a Low-Cost Optical Flow Sensor When Using an External Laser as a Direct Illumination Source. *Sensors (Basel, Switzerland)*, 11(12):11856–11870, December 2011.
- [12] Gao, Qingji, Wang, Yao, and Hu, Dandan. Onboard optical flow and vision based localization for a quadrotor in unstructured indoor environments. *IEEE Xplore*, January 2015.
- [13] S. S. Ghidary, T. Tani, T. Takamori, and M. Hattori. A new home robot positioning system (HRPS) using IR switched multi ultrasonic sensors. In *1999 IEEE International Conference on Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings*, volume 4, pages 737–741 vol.4, 1999.
- [14] Christoph Hertzberg, Rene Wagner, Udo Frese, and Lutz Schroder. Integrating Generic Sensor Fusion Algorithms with Sound State Representations through Encapsulation of Manifolds. *Spatial Cognition. Reasoning, Action, InteractionDeutsches Forschungszentrum fur Kunstliche Intelligenz (DFKI)*, July 2011.
- [15] Jinwook Huh, Woong Sik Chung, Sang Yep Nam, and Wan Kyun Chung. Mobile Robot Exploration in Indoor Environment Using Topological Structure with Invisible Barcodes. *ETRI Journal*, 29(2):189–200, April 2007.

- [16] Hwymers. An example factor graph, November 2008.
- [17] Kauai Labs Inc. Video Processing Latency Correction Algorithm, 2017.
- [18] Itseez. OpenCV: Calibration with ArUco and ChArUco, August 2017.
- [19] Itseez. OpenCV, Optical Flow, 2017.
- [20] Marcoe Keith. LIDAR an Introduction and Overview, 2007.
- [21] Hong-Shik Kim and Jong-Suk Choi. Advanced indoor localization using ultrasonic sensor and digital compass. In *2008 International Conference on Control, Automation and Systems*, pages 223–226, October 2008.
- [22] L. Kleeman. Optimal estimation of position and heading for mobile robots using ultrasonic beacons and dead-reckoning. In *Proceedings 1992 IEEE International Conference on Robotics and Automation*, pages 2582–2587 vol.3, May 1992.
- [23] Dongkyu Lee, Sangchul Lee, Sanghyuk Park, and Sangho Ko. Test and error parameter estimation for MEMS — based low cost IMU calibration. *International Journal of Precision Engineering and Manufacturing*, 12(4):597–603, August 2011.
- [24] J. J. Leonard and H. F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, June 1991.
- [25] Yangming Li and Edwin B. Olson. Extracting general-purpose features from LIDAR data. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1388–1393. IEEE, 2010.
- [26] Weiguo Lin, Songmin Jia, T. Abe, and K. Takase. Localization of mobile robot based on ID tag and WEB camera. In *IEEE Conference on Robotics, Automation and Mechatronics, 2004.*, volume 2, pages 851–856 vol.2, December 2004.
- [27] H. Liu, H. Darabi, P. Banerjee, and J. Liu. Survey of Wireless Indoor Positioning Techniques and Systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(6):1067–1080, November 2007.
- [28] Todd Lupton and Salah Sukkarieh. Visual-Inertial-Aided Navigation for High-Dynamic Motion in Built Environments Without Initial Conditions. *IEEE Press*, 28:61–76, February 2012.
- [29] Alvin Marquez, Brinda Tank, Sunil Kumar Meghani, Ahmed Ahmed, and Kemal Tepe. Accurate UWB and IMU based Indoor Localization for Autonomous Robots. In *IEEE 30th Canadian Conference on Electrical and Computer Engineering*, 2017.
- [30] F. M. Mirzaei and S. I. Roumeliotis. A Kalman Filter-Based Algorithm for IMU-Camera Calibration: Observability Analysis and Performance Evaluation. *IEEE Transactions on Robotics*, 24(5):1143–1156, October 2008.
- [31] Muñoz-Salinas, Rafael, Marín-Jiménez, Manuel, Yeguas-Bolívar, Enrique, and Medina-Carnicer, Rafael. Mapping and Localization from Planar Markers. *Pattern Recognition*, 2016.

- [32] NASA. Kalman Filter Integration of Modern Guidance and Navigation T1c Systems, 1999.
- [33] Peter O'Donovan. Optical Flow: Techniques and Application, April 2005.
- [34] Open Source Computer Vision. Detection of ArUco Markers, December 2015.
- [35] Pozyx. Pozyx - centimeter positioning for Arduino, 2017.
- [36] Richard Hartley and Andrew Zisserman. Multiple View Geometry in Computer Vision, 2003. p312.
- [37] S. S. Saab and Z. S. Nakad. A Standalone RFID Indoor Positioning System Using Passive Tags. *IEEE Transactions on Industrial Electronics*, 58(5):1961–1970, May 2011.
- [38] T. Sattler, B. Leibe, and L. Kobbelt. Fast image-based localization using direct 2d-to-3d matching. In *2011 International Conference on Computer Vision*, pages 667–674, November 2011.
- [39] A. Schlichting and C. Brenner. VEHICLE LOCALIZATION BY LIDAR POINT CORRELATION IMPROVED BY CHANGE DETECTION. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLI-B1:703–710, June 2016.
- [40] Adam Smith, Hari Balakrishnan, Michel Goraczko, and Nissanka Priyantha. Tracking Moving Devices with the Cricket Location System. In *Proceedings of the 2Nd International Conference on Mobile Systems, Applications, and Services*, MobiSys '04, pages 190–202, New York, NY, USA, 2004. ACM.
- [41] Min Sun. Optical Flow, 2008.
- [42] Juan Tardos, José Neira, Paul M. Newman, and John J. Leonard. Robust Mapping and Localization in Indoor Environments Using Sonar Data. *The International Journal of Robotics Research*, 21, April 2002.
- [43] D. Tedaldi, A. Pretto, and E. Menegatti. A robust and easy to implement method for IMU calibration without external equipments. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3042–3049, May 2014.
- [44] Luka Teslic, Igor Skrjanc, and Gregor Klancar. EKF-Based Localization of a Wheeled Mobile Robot in Structured Environments. *Journal of Intelligent and Robotic Systems*, May 2011.
- [45] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.
- [46] Lidar UK. How does LiDAR Work?, 2017.
- [47] Vadim Indelman, Stephen Williams, Michael Kaess, and Frank Dellaert. Information Fusion in Navigation Systems via Factor Graph Based Incremental Smoothing. *Robotics and Autonomous Systems*, 61(8):721 – 738, 2013.

- [48] John Wang and Edwin Olson. AprilTag 2: Efficient and robust fiducial detection. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4193–4198. IEEE, 2016.
- [49] A. Ward, A. Jones, and A. Hopper. A new location technique for the active office. *IEEE Personal Communications*, 4(5):42–47, October 1997.
- [50] W. Xu and S. McCloskey. 2d Barcode localization and motion deblurring using a flutter shutter camera. In *2011 IEEE Workshop on Applications of Computer Vision (WACV)*, pages 159–165, January 2011.
- [51] H. Yucel, R. Edizkan, T. Ozkir, and A. Yazici. Development of indoor positioning system with ultrasonic and infrared signals. In *2012 International Symposium on Innovations in Intelligent Systems and Applications*, pages 1–4, July 2012.
- [52] Zebra. Dart Ultra Wideband UWB Technology | Zebra, 2017.