# Localization for FRC

C Term Report

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

February 18, 2018

## 1 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 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

*equal contribution from all authors

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 Related Work section. The strengths and weaknesses of these existing methods are described in the Evaluation of Localization Techniques 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.

# 2 Related Work

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.

## 2.1 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 [18]. Light travels at about $0.3\,\mathrm{m\,ns^{-1}}$ which is a constant speed [18]. 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 [18]. 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 [18]. 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 [18]. 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 [18]. The wavelength for these is typically 1.0-1.5 m [43]. The second system uses high energy lasers and is typically only used for atmospheric measurements [18]. The wavelength of these is typically 0.5-0.55 m [43]. 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 [36]. 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 [23].

## 2.2 Ultrasonic Mapping

Ultrasonic mapping (often referred to as sonar) was one of the first techniques used for indoor robot localization, and has been explored deeply since the 1980's. The most common approach is to use multiple emitter-receiver transducers placed around the perimeter of the robot, measure the range at each of those points, then localized to a given map of the environment [10]. Alternately, some systems use one sensor and rotate it around to achieve the same effect [22, 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 [22], 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 [39], in which 24 sonar sensors in conjunction with encoders were used to perform simultaneous localization and mapping. Their experimental results report drifts of 3.9 m and 21° over the course of 35 m of travel.

## 2.3 IMUs and Encoders

An inertial measurement unit (IMU) is a sensor reporting the forces acting upon an object, the angular rates of change, and surrounding magnetic field. The device typically comprises an accelerometer, gyroscope, and magnetometer which sense the above data respectively. These devices function by detecting Coriolis forces, inertial forces acting in the direction of motion relative to a rotating reference frame. These forces are proportional to the magnitude of the acceleration. These forces may be detected by a simple strain gauge mechanism or object vibrating at a known frequency (the rate of change of vibration frequency is detected) [3]. The premise behind position sensing using this device involves integrating the data with respect to time to calculate position and orientation. This approach was first used in aeronautics to estimate projectile attitude, orientation, and position [30]. 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 [3].

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 [13]. 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 [41] or indoor positioning systems that use radio frequency signals [27].

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 [5].

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 [26]. 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 [44]. 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.

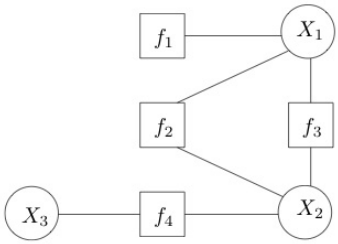



Figure 1: A simple factor graph [15].

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 [16]. 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.

## 2.4 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 [2] 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 [9] uses passive RFID tags on the ceiling and an RFID transmitter on the robot, and report an accuracy of 4cm within a $5m^2$. Another RFID system [34] 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 [25]. 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 [49] [32].

Among ultrasonic beacon systems, [20] 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 [37], [46], and [19] 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 [12] [48]. These systems are inexpensive, and report accuracy of between 2 cm and 14 cm.

## 2.5 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 [35] 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 [24] 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 $20\,\mathrm{cm\,s^{-1}}$. This system measured the position and orientation of the robot relative to the ID tags through image processing. The result of the experiment in [24] showed that this method was accurate even though there was an unevenness between the ceiling and the floor. Another system [14] 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) [8] used 1d barcodes as references. Using a camera with 80 vertical pixels and 640 horizontal pixels,

the system achieved localization within $3.5\,\mathrm{m\,s^{-1}}$ 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 $50\,\mathrm{mm\,s^{-1}}$, 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.

## 2.6 Optical Flow

Optical flow is the ability to track changes between cameras frames and measure the differences between them to track position. In other words, optical flow is 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[31].

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 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 [38]. $\alpha$ 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 [31]. $n$ 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.

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

Figure 2: Horn and Schunk Optical Flow vector equation

Optical flow can also be done locally using the Lucas Kanade method [38]. 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 [31].

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

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 [31].

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 [38].

## 2.7 Filtering and Calibration

Given the number of sources of position information, it is natural that there will also be a number of ways to take advantage of using multiple techniques together. Different sensors can have better or worse performance in different scenarios, and a choice of fusion algorithm will yield more accurate position information by leveraging this. Calibration can also be used to compensate for errors between sensors. For instance, if your 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, where as particle filters can more be more accurate if the true belief distribution is non-gaussian or if the true dynamics are nonlinear [42]. 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 now 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 gaurantees and their efficacy has been demonstrated in numerous systems [6][9][28][30][34].

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 calcuating these calibration parameters by taking advantage of static intervals and assumptions about the force of gravity [26][21][40]. Visual tag detection algorithms, such as ArUco, also include a camera calibration process to account for the focal length, field of view, and distortion characterstics of the camera [1]. Knowing these parameters allows one to undo distortion to the image, which is essential for detection of most AR tags.

# 3 Evaluation of Localization Techniques

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 [19] 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 2, techniques for localization include LIDAR mapping, ultrasonic mapping, IMU and encoders, infrared or radio and ultrasonic beacons, wireless network methods, cameras with tags, and optical flow. Each of these techniques has been used successfully in their respective applications, but not all of them are appropriate for this project.

LIDAR has been shown to be one of the highest performing localization methods in terms of accuracy, precision and update rate. The two reasons why we are not pursuing it further are because it is too expensive and because it requires a map. LIDARs capable of ranging across an entire FRC field are over $400, which is the cost limit for any single part on an FRC robot. Additionally, LIDAR techniques also require either mapping on the fly, or an existing map. Mapping on the fly presents its own challenges, and usually suffers from very bad localization for some initial period of time while the map is built. Therefore, a map would have to be provided for the environment. Existing maps would work very well on the competition FRC fields, but would not apply in the practice spaces teams 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 5.3, are projected to cost about $30 (see 3). 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 5.5.4. 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 contains 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 [45] 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° off) when the off - axis angle was small, but were within 1 degree from 20° to 75° of off - axis angle. The detected rates for tags were 100% from 0 to 17 m away. This system showed that the combination of camera and fiducial tags can potentially localize robots accurately and precisely. The research, [4] 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 [47] 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[29].

| *Sequence* | Ours | LSD-SLAM | ORB-SLAM2 |
|---|---|---|---|
| SLAM-Seq 1 | **0.0447** | 0.440 | 0.231 |
| SLAM-Seq 2 | **0.0433** | 0.117 | 0.913 |
| SLAM-Seq 3 | **0.0694** | 0.652 | 0.314 |

Figure 4: 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 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 [38]. 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 measure over rotation in order to navigate a computer screen. Lighting is also important to for the camera to be able to clearly pick up images so having a source of light illuminating around the optical flow mouse would also be necessary for teams in order to get the best results [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 [17]. 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.

# 4 Experimental Results

## 4.1 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 5 shows the fully assembled robot.

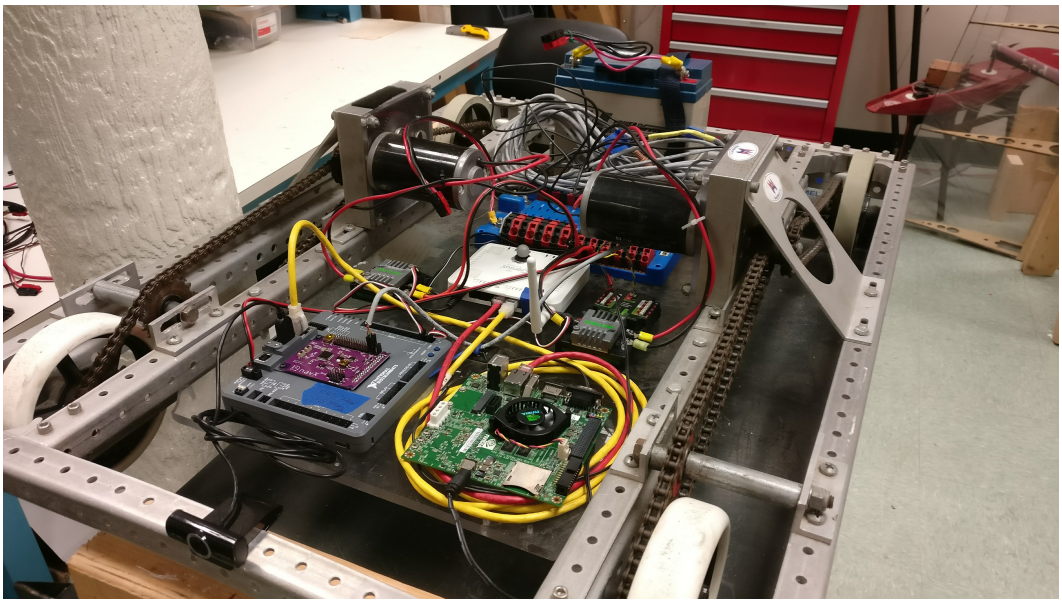


Figure 5: Test Robot used in our experiments

## 4.2 Double Integration of Accelerometer

Physics and calculus tell us that we can recover position information by double integrating acceleration. As a baseline, we wanted to see how well simply double integrating our accelerometer works. We drove a Turtlebot with a NavX around on a carpet surface and logged the encoder and accelerometer readings at 10 Hz. We then double integrated the encoder data (We used Equation 3 to perform the integration).

$$\begin{aligned} v^x_{t+1} &= v^x_t + a^x_t \Delta t \\ v^y_{t+1} &= v^y_t + a^y_t \Delta t \\ p^x_{t+1} &= p^x_t + v^x_t \Delta t + 0.5 a^x_t \Delta t^2 \\ p^y_{t+1} &= p^y_t + v^y_t \Delta t + 0.5 a^y_t \Delta t^2 \end{aligned} \tag{3}$$

Equation 3: $a^x_t$ is the measured acceleration in the $x$ axis at time t.

We compared the resulting position to the position derived from forward kinematics of the encoder values. Because the Turtlebot is differential drive, moves slowly, and was driven on carpet, there is no slip, and so the position derived from encoders is extremely accuracy. Figure 6 shows how adding a simple bias and scaling factor can improve the accurate of double integration.

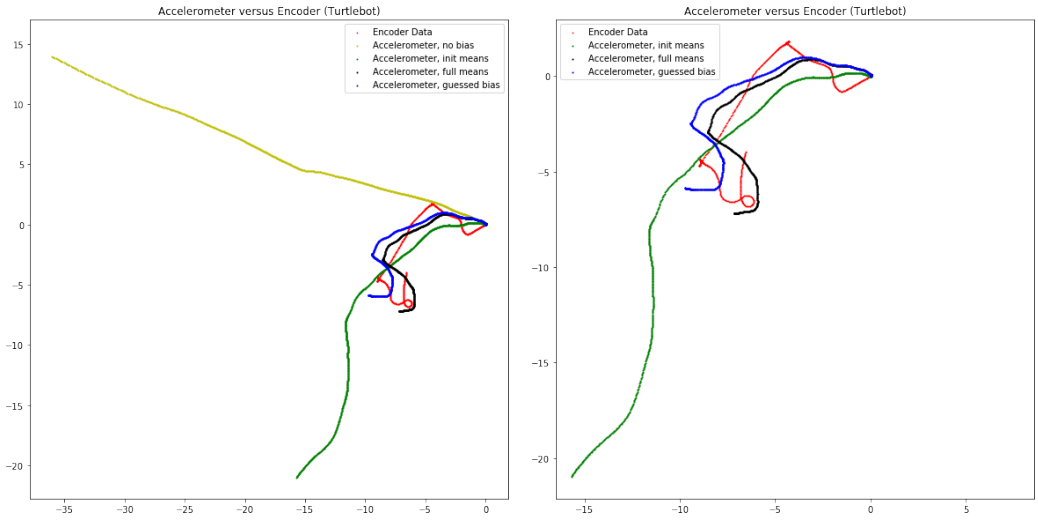



Figure 6: Comparison of unbiased double integration with various biases

This demonstrated clearly that there are some biases for which the accelerometer data, when double integrated, gives a useful estimate of position. The question then becomes solving for these constants. In the figures above, the green line uses biases that were computed by taking the average accelerometer readings over the first 40 readings, where the robot was not moving. This may have improved the results, but not significantly. The black line used averages over all the data (including while the robot is driving), which is suprisingly more accurate. The blue line uses hand-guesses biases, and is included to demonstrate that it is possible to find alternate biases that also do well.

## 4.3 IMU Calibration

From an early experiment collecting data on a Turtlebot, we saw that double integrating the accelerometer readings would not be accurate. This emperical observation is in line with the literature. Therefore, we replicated the IMU calibration procedure in [40], which accounts for many sources of error without requiring expensive external equipment. This calibration method was relatively to perform, and could be replicated by the FRC students. This method corrects the misalignment, scaling, and biases in both accelerometer and gyroscope. Generally, 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 closest to the true angles. First, the IMU needed to be placed statically for a period of $T_init$ seconds. $T_init$ can be calculated by Allan variance. 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 proecdure 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.

We modified the method used in [40] to cater our situation. During our IMU calibration period, we used 45 s for $T_init$ as suggested in [40] instead of calculating the Allan variance. In order to get as many distinct positions as possible, a Helping-Hands was used to hold the IMU. We adjusted the Helping-Hands after the IMU was static for 4 seconds. We rotated the IMU 36 times in total. The accelerometer data and gyroscope data in x, y, z axis were recording for the entire period. Using the threshold from initialization data and the full accelerometer data, the static detector successfully distinguished between static intervals and dynamic intervals. A demonstration of our static detector is shown in Figure 7.

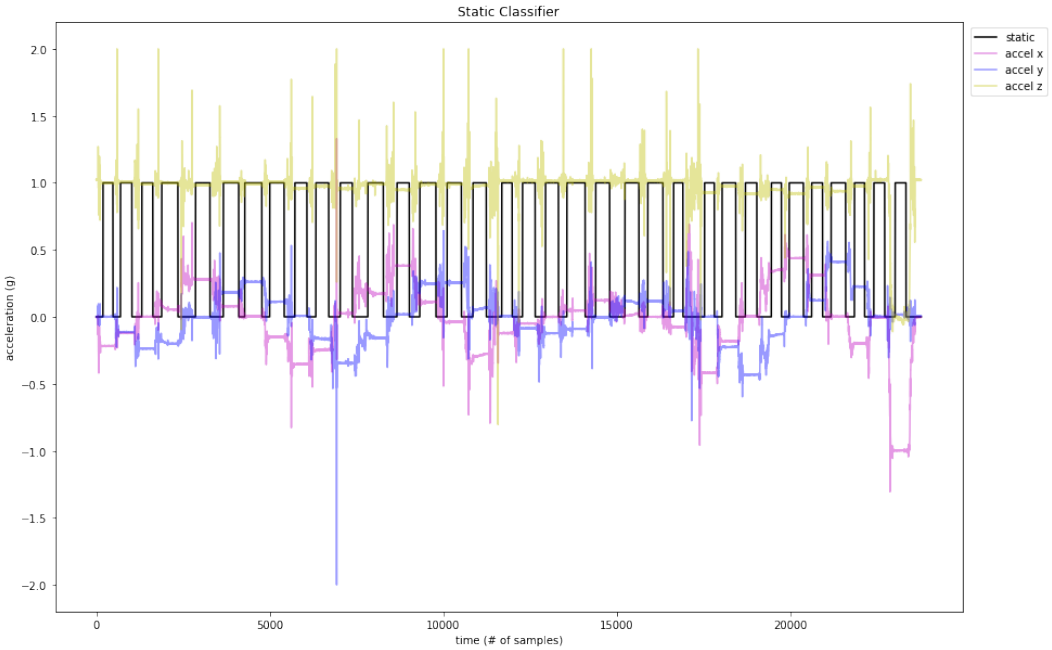



Figure 7: The black line is 1 during intervals classified as static

Using the identified static intervals, we optimize using the Levenburg-Marquedt procedure in python's NumPy package to solve for the accelerometer calibration values. The equation we are minimizing is shown below (Equation 4). These values can be found in Table 1, and descriptions of each variable can be found in [40].

$$\|g\|^2 - \|T^a K^a (a^s + b^a)\|^2 \tag{4}$$

| $\alpha_{yz}$ | $\alpha_{zy}$ | $\alpha_{zx}$ | $s_x^a$ | $s_x^a$ | $s_z^a$ | $b_x^a$ | $b_y^a$ | $b_z^a$ |
|---|---|---|---|---|---|---|---|---|
| -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 transormation, $[0, 0, 0, 1, 1, 1, 0, 0, 0]$. 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 data closes to the true orientation. This is expressed in the error function we are minimizing, shown in Equation 5

$$\begin{gathered} \left\| u_{a,k+1} - \int_{k}^{k+1} \Omega(\omega_i^S) u_{a,k} di \right\| \\ \Omega(\omega_i^S) = T^g K^g (\omega_i^S + b^g) \end{gathered} \tag{5}$$

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 integrad, rather than a analytic closed-form. Therefore, numerical integration methods like Euler's Forwardmethod or Runga-Kutta methods can be used. While [40] uses Runga-Kutta 4th Order (RK4), we start with the simpler Euler's Forward method, but we intend to test with RK4 as well. The function $\Omega(\omega_i^S)$ takes the raw angular velocity readings $w_i^S$, 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 $u_a$. Over the whole integral, this rotates the average acceleration values from the $k$th static interval, $u_{a,k}$, to the average acceleration values from the $k+1$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 mitagates the issues of using euler-angle rotation matrices. This theoretical result was tested experimentally, and the results are shown in Figure 8. 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.

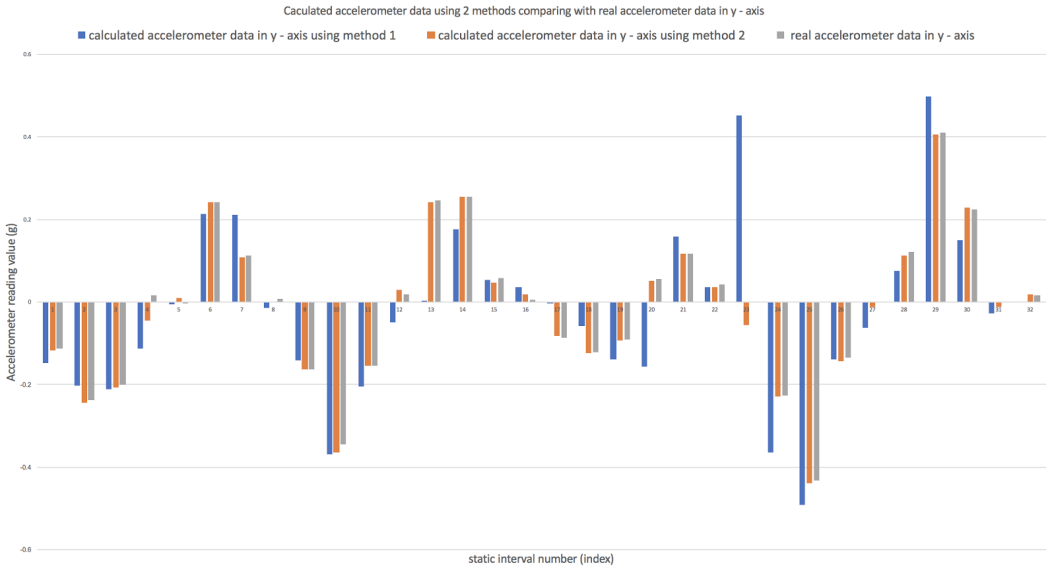



Figure 8: Integration of the gyroscope readings in the Y Axis. Method 1 is one-shot, Method 2 is incremental rotation.

## 4.4 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 19 illustrates the various delays we need to account for. We conducted experiments to get initial estimates of these delays. First, to get an estimate of the radio transmit receive delay, a transmitter and receiver were set up on two microcontrollers. The transmitter sent 5 ms pulses of radio energy (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 from the total time the theoretical time of flight of the radio signal. The full data for these measurements are available in Appendix B, and an example measurement is shown in Figure 9. The time of flight of radio over distances of a new centimeters or meters is on the order of nanoseconds. We measured an average delay of 45.175 µs, which we attribute to the internal circuitry of the transmitter and receiver. The variance of this delay was 16 µs. However, we also measured delays as low as 32 µs and as high as 79 µs. 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 information will be used to better model the timing of our beacons to make the system as accurate as possible.

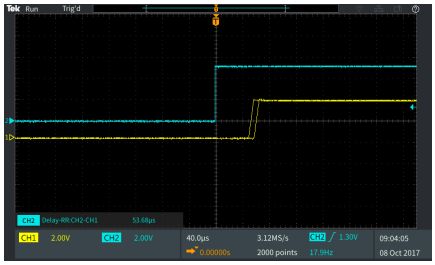



Figure 9: Example measurement total trip time for radio signal. The blue line is the input to the transmitter, and the yellow is the output of the receiver

Next we performed a similar experiment with the ultrasonic transducers. For this experiment, we used two NTX-1004PZ piezo tweeters placed 25 cm apart. The NTX-1004PZ is meant to be a high-frequency speaker for DJ equipment, and is designed to operate between 4 kHz and 20 kHz. However, because they are incredibly cheap we decided to evaluate them as ultrasonic speakers running just above that range. One was connected to a PSoC 5LP for transmitting, and the other was connected only to the oscilloscope. The other oscilloscope prove was connected to the transmitting piezo. The time difference between the transmitting signal on and the receiving signal was measured. The signal applied to the transmitter was short bursts of a 24kHz square wave. The measure time delay between the transmit and receive very closely matched the expected time for sound to travel between the speakers. This indicates that we will not need to account for constant delays in the transmit or receive circuits. However, there were several other noteworthy behaviors we discovered during these tests. First, we noticed that any immediate changes in the frequency of the transmit signal will clicks in the audible range. these clicks do not effect the distance measurement, but they are mildly annoying and should be suppressed if possible by changing the transmit signal.

## 4.5 Measuring Frequency Response

After testing for delays in the ultrasonic circuitry, we also measured the frequency response of the NTX-1004PZ. We placed two tweeters in the same configuration as described above. 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 10 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.

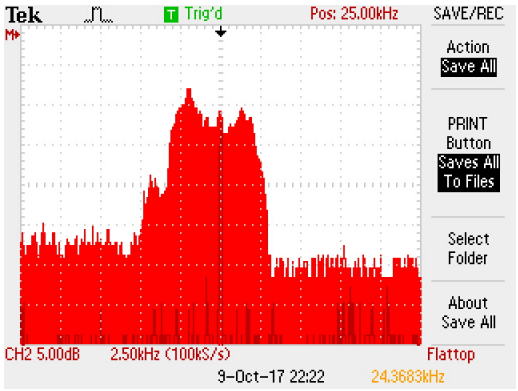



Figure 10: Frequency response of the NTX-1004PZ. The best response is achieved at 22kHz, and the highest detectable frequency is 27kHz.

## 4.6 Optical Flow

Preliminary testing has been done using a Microsoft USB camera using the sample code provided for dense optical flow in OpenCV. In the screenshot below it can be see in 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.

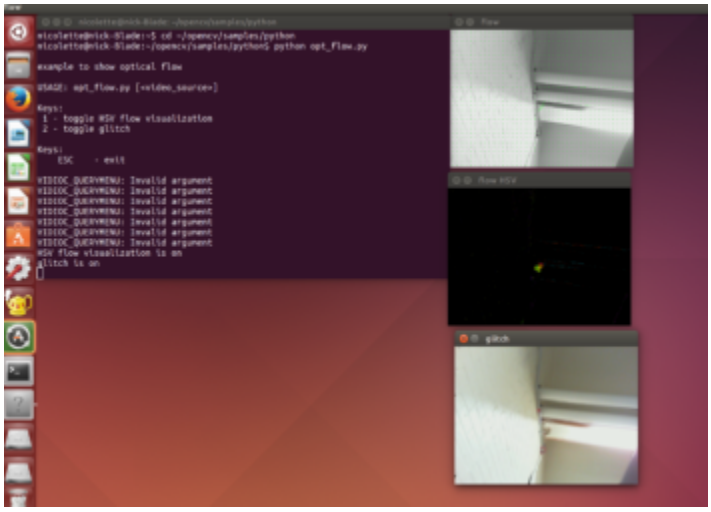


Figure 11: Screenshot of optflow running from OpenCV samples

Testing was then done with the demo for Lucas Kanade based optical flow. Tests were done using lkdemo.cpp which was a sample file provided by OpenCV. Tests were done to

see how running this program on a laptop verse the RoboRIO and compare the time they took to run the code. The laptop that the code was run on has a 2.8 GHz Intel 4 Core i7 processor for reference. A chart below was made of the time that each program took to run 100 frames in seconds.

| | Laptop (s) | RoboRIO (s) |
|---|---|---|
| | 3.638 | 8.429 |
| | 4.184 | 8.429 |
| | 3.638 | 8.429 |
| | 3.639 | 8.429 |
| | 4.184 | 8.429 |
| Average | 26 | 12 |

Table 2: Time for 100 frames to run using OpenCV on laptop verse RoboRIO

After measuring each 5 times to check for repeatability it can be seen looking at the chart except for the last row how fast in seconds it took for each of the devices to run 100 frames. The last row frames per second was calculated for each device with the laptop reading approximately 26 FPS and the RoboRIO reading 12 FPS. While the laptop was over twice that of the RoboRIO the 12 FPS rate is still fast enough to meet the needs of the project. Other preliminary tests were done with shaking the camera as the program was running after having it initialize the image with marked pixels it had deemed best for measuring. After some shaking of the camera most marked pixels had stayed and only ones that were causes to go off screen by the shaking disappeared. The pixels that the program had decided to measure were based off of what types of pixels work best for Lucas Kanade base optical flow described in the research section of the paper .

## 4.7 Camera and ArUco Tag

We performed an experiment on clarity of ArUco tags with respect to textures and distance using ArUco tags with a 720p web camera. Six ArUco codes with different IDs were generated by an online ArUco tag generator. These tags have 3 different sizes: 100 mm, 150 mm, 195 mm and 2 different textures: matte, reflective. Those tags were taped together on plywood.

A program using OpenCV ArUco library was ran for 100 frames to count the times of different tags were recognized. For each distance, the program was ran 3 times and the average of the shown up times was calculated. The results of the experiments were shown in the diagram below (Figure 12).

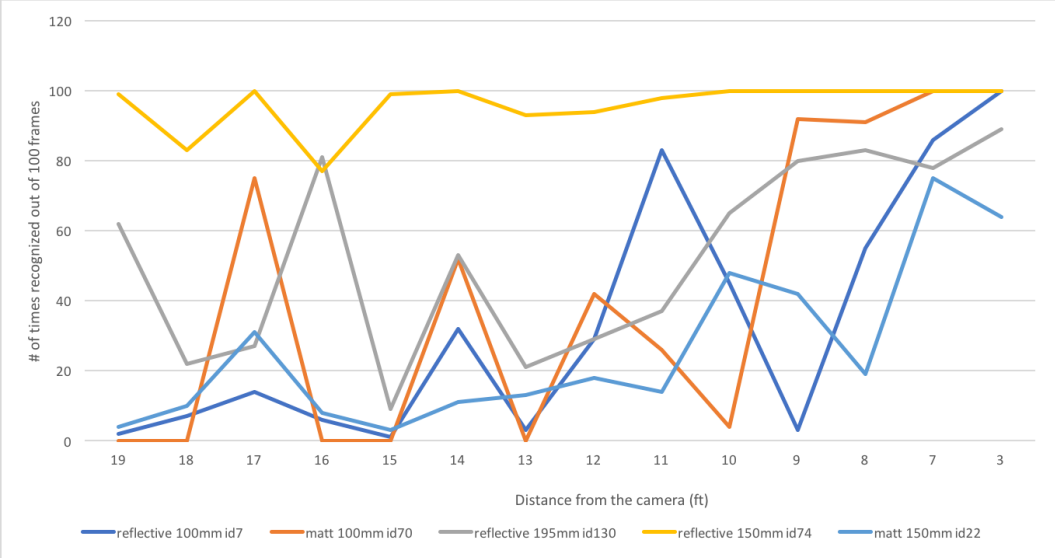



Figure 12: Clarity of ArUco tags experiment

There must be other factors other than textures and distances significantly changing the result of this experiment. A similar but improved experiment will be done in the future.

### 4.7.1 Localization Algorithm

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.

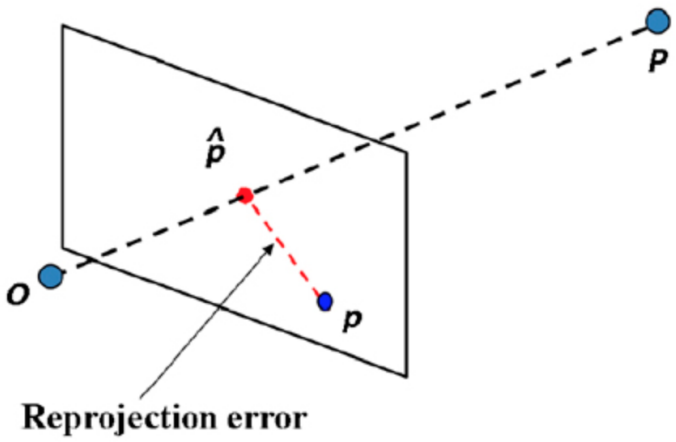


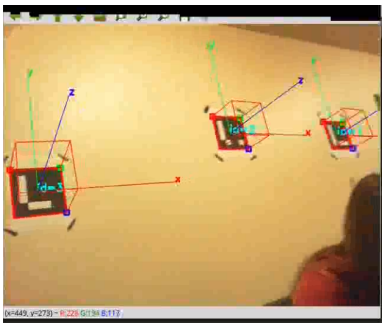

Figure 13: Calculating reprojection error [33].

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.

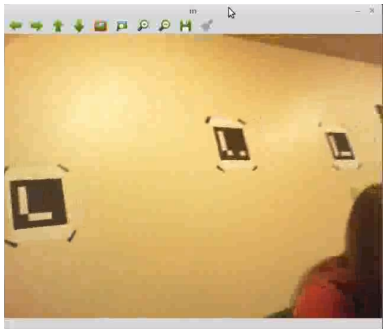
$$p = \begin{bmatrix} t \\ r \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \\ t_z \\ r_x \\ r_y \\ r_z \end{bmatrix} \tag{6}$$

$$T_0^1 = p_1 - p_0 \tag{7}$$

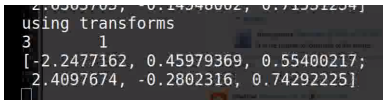
The map is complete when the user is able to query the pose from the origin using data from any tag in the workspace. Previous research indicates that calculating the transform between tags is sufficiently accurate. This protocol is preferred because it does not require the user to manually measure distances 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.



(a) annotated frame



(b) raw data




(c) terminal output

Figure 14: Visualization of pose estimate

## 4.8 Kalman Filtering

We designed a Kalman filter that combines measured and predicted data to find the optimal estimate of robot position. The dynamics of the robot had to be calculated to find the state from measured sensor data and a model had to be made of the robot.The state variable $x$ describes what variables are going to be measured and predicted. We decided that the state needs to represent the position, velocity, and acceleration of the robot which can be described using $x$, $y$, $\theta$, $\dot{x}$, $\dot{y}$, $\dot{\theta}$, $\ddot{x}$, $\ddot{y}$, and $\ddot{\theta}$. The control inputs which will be $u$ will be acceleration of the left wheel and right wheel. The sensors contributing data to the filter are an accelerometer, gyroscope, encoders, and camera. The Kalman filter consists of two sets of equations. The first equation predicts the current state by using the state estimate from the previous time step and the current input which can be seen in equation 8 below.

$$\hat{x}_t = Ax_t + Bu_t \tag{8}$$

The A matrix describes the robots dynamics trying to predict the next state given the current state. The dynamics equations are used to calculate final position,velocity,and acceleration for given current position, velocity, and acceleration. The equations for prediction for current position in $x$ and current velocity in $x$ are shown below in equation 9.

$$\begin{aligned} x_{t+1} &= x_t + \dot{x}_t \Delta t + \tfrac{1}{2}\ddot{x}_t \Delta t^2 \\ \dot{x}_{t+1} &= \dot{x}_t + \ddot{x}_t \Delta t \end{aligned} \tag{9}$$

This estimation was done for each variable in the state vector $x$ for position,velocity, and acceleration and put into matrix multiplication form. Below in matrix 10 is the final A dynamics matrix and $x$ state vector.

$$\begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}\Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}\Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}\Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{bmatrix} \tag{10}$$

The B matrix describes the change in state given the inputs to the motors. This matrix derives its equations for change in position and velocity estimation based off of equations for finding the acceleration based off the left and right wheel velocities. For acceleration in the $x$ and $y$ directions the average of the accelerations in left and right wheels and then breaks it into $x$ and $y$ components by multiplying by $\cos(\theta)$ or $\sin(\theta)$. The equation used to find theta for acceleration can be seen in equation 11 below with R being the wheel radius while W track width between wheels.

$$\begin{aligned} \Delta x_{t+1} &= \cos(\theta)\frac{1}{4}\Delta t^2\alpha_l + \cos(\theta)\frac{1}{4}\Delta t^2\alpha_r \\ \Delta y_{t+1} &= \sin(\theta)\frac{1}{4}\Delta t^2\alpha_l + \sin(\theta)\frac{1}{4}\Delta t^2\alpha_r \\ \Delta\theta_{t+1} &= \frac{R\Delta t^2}{W2}\alpha_l + \frac{-R\Delta t^2}{W2}\alpha_r \\ \Delta\dot{x}_{t+1} &= \cos(\theta_t)\frac{\alpha_l + \alpha_r}{2}\Delta t \\ \Delta\dot{y}_{t+1} &= \sin(\theta_t)\frac{\alpha_l + \alpha_r}{2}\Delta t \\ \Delta\dot{\theta}_{t+1} &= \frac{R\Delta t}{W}\alpha_l + \frac{-R\Delta t}{W}\alpha_r \end{aligned} \tag{11}$$

The prediction of change in state for acceleration was calculated using these equations. The velocity equations were found by integrating with respect to time. For position the equations for the A matrix position since they had included acceleration already. The final B matrix and $u$ vector can be seen below in equation 12. The predicted state measurement can then be calculated by following the equation shown in equation 8 .

$$\begin{bmatrix} \cos(\theta_t)\frac{1}{4}\Delta t^2 & \cos(\theta_t)\frac{1}{4}\Delta t^2 \\ \sin(\theta_t)\frac{1}{4}\Delta t^2 & \sin(\theta_t)\frac{1}{4}\Delta t^2 \\ \frac{R\Delta t^2}{2W} & \frac{-R\Delta t^2}{2W} \\ \cos(\theta_t)\frac{1}{2}\Delta t & \cos(\theta_t)\frac{1}{2}\Delta t \\ \sin(\theta_t)\frac{1}{2}\Delta t & \sin(\theta_t)\frac{1}{2}\Delta t \\ \frac{R\Delta t}{W} & \frac{-R\Delta t}{W} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \alpha_l \\ \alpha_r \end{bmatrix} \tag{12}$$

Predicting the error covariance for the next state was was the next step. The error covariance represented by P is the variance in the estimate found by the equation in equation

13.The equation to calculate is is shown below in equation 13. The variable P is the estimate covariance matrix while Q is the process covariance matrix shown below in equation 13. The estimate covariance matrix is the covariance between the estimated error covariance before and after the measurements were updated. The process covariance matrix is a matrix estimating the process noise which we found to be a diagonal matrix.

$$P_{t+1} = AP_tA^T + Q \tag{13}$$

The next step in Kalman filtering is updating the measured values and finding the new estimate of the state based off the updated values. The update equations are shown below in equation 14.

$$\begin{aligned} K_t &= P_tC^T(CP_tC^T + R)^{-1} \\ x &= \hat{x} + K(z_t - C\hat{x}_t) \\ P_t &= (I - K_tC)P_t \end{aligned} \tag{14}$$

The first equation that calculates K minimizes the estimation of errors variance. The equation finds the ratio of the estimate covariance(P) and the estimate covariance plus the measurement variance(R). Whichever estimate has less error will weigh more towards the final estimation of state shown in the middle equation. To estimate x based off of updated measurements the second equation shown in the list of equations 14 is used. The measurements that are being recorded from each sensor are stated in measurement matrix $z$. The matrix C if a matrix of ones that hold the place of where each measurement is in the 9 by 9 matrix so they can be subtracted from $z$. Matrix C is then multiplied by the state vector and then subtracted from $z$ as shown below in equation 15. The difference between the updated measurements and the estimates of the state previously are found. That value is what is then adjusted by K. That value is then added to the a priori estimate to get the updated estimate of $x$. A similar equation is used to calculate the new error covariance. The variable I is a matrix of 1s on the diagonal. The K value is multiplied by C and then subtracted from I so each value in the I matrix is set based off of the value of K.

$$\begin{bmatrix} \text{Accelerometer } \ddot{x} \\ \text{Accelerometer } \ddot{y} \\ \text{Gyro } \dot{\theta} \\ \text{Camera } x \\ \text{Camera } y \\ \text{Camera } \theta \\ \text{Encoders } x \\ \text{Encoders } y \\ \text{Encoders } \theta \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{bmatrix} \tag{15}$$

# 5 System Specification

## 5.1 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 our 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.

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 know that most FRC shooting and pickup mechanisms will work within $\pm 10\,\text{cm}$. 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 $\pm 5\,\text{cm}$ 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 $6\,\text{m}\,\text{s}^{-1}$, 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.

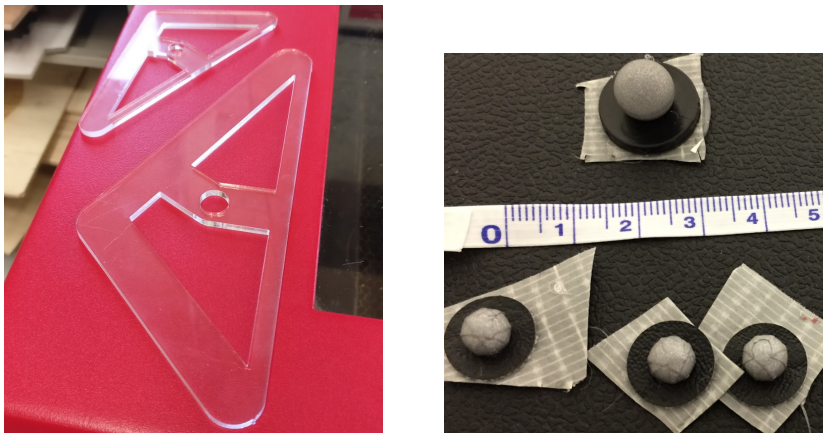
Using all of these informal estimates as a starting point, we conducted a survey of FRC students, Alumni, and mentors. We received 65 responses in total, and used the results of this survey to solidify our design criteria. The full response of this survey are presented in Appendix A. In summary, the median for accuracy was 10 cm in x,y and 5° 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 $ft^2$, 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.

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.

## 5.2 Vicon Test

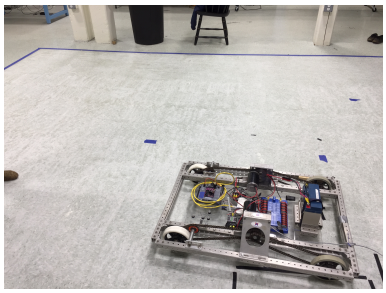
The ground truth data for measuring accuracy and precision is obtained using a Vicon brand Motion Capture system. This comprises a Vicon Lock+ data processor and 8 Vero infrared cameras. These can collect 2.2 megapixels of data at maximally 330 FPS and are designed for capturing human motion in small spaces. The space used for experimentation was 19x14 feet. The pose of the robot is tracking 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.



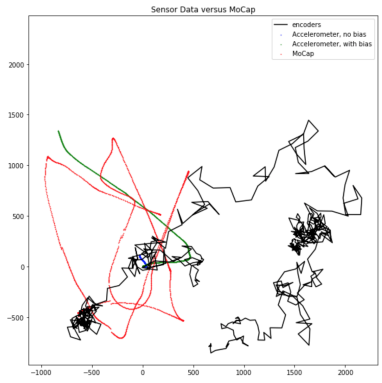

(a) guide for placement (b) reflective markers

Figure 15: Vicon tracker set up

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 Jetson TK1. This data is synchronous to within a few (1-2) milliseconds. Using the same markers, the pose of the AruCo tags is also measured. Preliminary testing suggests that the odometry and IMU datasets alone are insufficient for position tracking when the robot is prone to skidding. Camera data has not been processed, so results are omitted.



(a) robot in Vicon field




(b) IMU, encoder, and Vicon position data

Figure 16: Collecting and Plotting Position data

Because the Vicon system is meant for high-resolution tracking in a small area, a robot with a large footprint and wheels prone to skidding may be inappropriate for early tests. Further testing is necessary using a smaller robot and a varying density of fiducial markers to obtain useful accuracy and precision data.

## 5.3 System Specification

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.

## 5.4 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 $3\,\mathrm{m\,s^{-1}}$.
- 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

These goals will hopefully constrain the scope of each localization technique such that all of them can be incorporated to some degree. Furthermore, we have drafted a timeline of which of these goals we will focus on over the course of the year. For example, by choosing to start with encoders, IMUs, and cameras, we will be able to have a working localization system even without the other techniques. Additionally, we hope that by giving an end date for some of the components we will be able to move on and not spend too long optimizing any one component.

- C Term:
  - Finish base-frame calibration of IMU
  - Meet goals for beacons
  - Decide between optical flow and IMU + encoders for dead reckoning
  - Filter each sensor together into one position update
- D Term:
  - Meet documentation goals for teams
  - Provide the designs and the code for all components
  - Write the final report

As of the end of B term, we have yet to demonstrate position information from the IMU and encoders on our test robot. We have shown position estimates from encoders are accurate when the wheels do not slip, but that is not realistic to the FRC environment. We have also separately shown that there exists some biases and scaling factors that allows us to double integrate our accelerometer readings and get a reasonable position estimate, but we have no demonstrated a successful methodology for computing these calibration values. Therefore, we are left with a choice to continue to work on IMU and encoders, or to use optical flow. Both would meet our requirements for high-speed local position updates, and would nicely compliment beacons and cameras with tags. In order to decide whether to explore optical flow or continue exploring IMU and encoders, we will return to existing research and determine which technique has performed better in other systems. If we can show that dead reckoning with IMU and encoders in an environment like FRC meets our system specification requirements, we will continue that method. If not, then we will pursue optical flow instead.

## 5.5 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.

### 5.5.1 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.

### 5.5.2 Communications and Latency

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 of UDP data between the RoboRIO and the TK1. We recorded the round trip time of these packets, which can be seen in Figure 17. The 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.

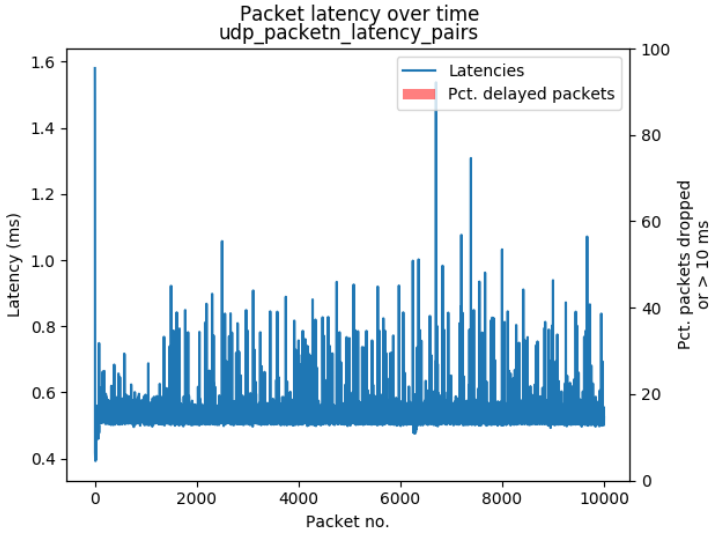



Figure 17: RTT of UDP packets between the RoboRIO and the TK1 over the robot's wired network.

Another importnat 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 [7]. 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 allos the sensor data sent from the RoboRIO to be synchronized with the clock on the TK1.

### 5.5.3 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. Inputing 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**

### 5.5.4 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.

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

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

This function will generate a chirp with amplitude $A$, starting from frequency $f_0$ going to $f_1$ over time interval $T$. Figure 18 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 19: Timing of radio and ultrasonic signals. Experiments indicate 46.175 µs total RF delay and 1 ms total ultrasonic delay.

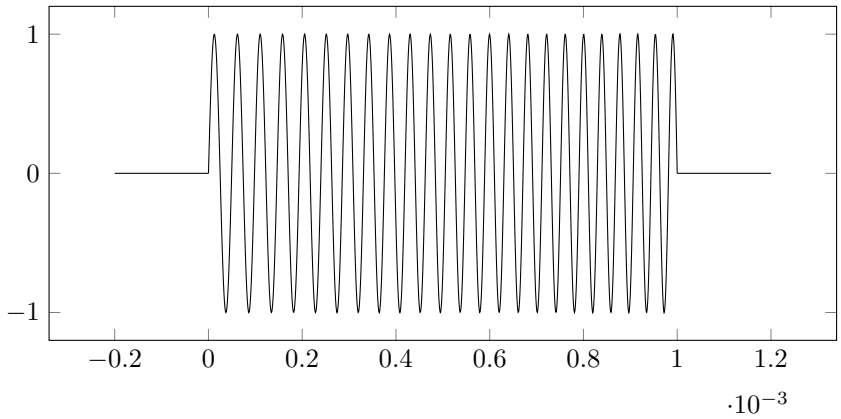



Figure 18: Chirp signal with $\phi = 0$, $A = 1$, $f_0 = 20\,\text{kHz}$, $f_1 = 27\,\text{kHz}$, and $T = 1\,\text{ms}$

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 18 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 19 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.

| 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 3: estimated bill of material for beacons

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

**Self-Localization of Beacons**

Beacons that can automatically discover each other and their relative positions will tremendously improve the easy 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.

#### 5.5.5 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.

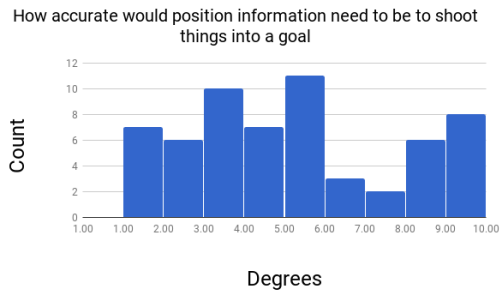
# 6 Conclusion

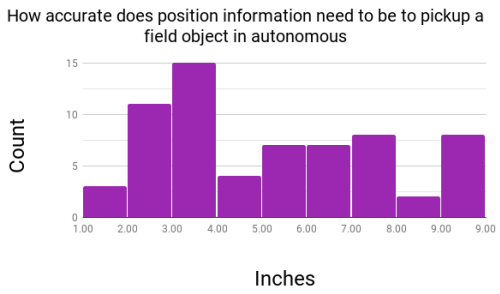
Over the course of A Term, we conducted thorough background research on various methods for localization and evaluated their performance. Based on numerous reports for each localization technique, a survey of FRC students and alumni, and our own knowledge of the challenges of localization for FRC we 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.

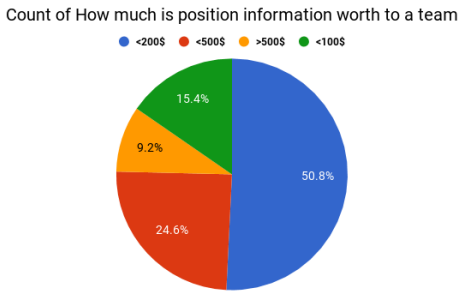
# 7 Appendix

## 7.1 Appendix A

Survey Responses




## 7.2 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 4: The time of flight of radio over tens of centimeters is insignificant compared to the delay within the transmitter and receiver.

# References

[1] OpenCV: Calibration with ArUco and ChArUco, August 2017.

[2] P. Bahl and V. N. Padmanabhan. RADAR: an in-building RF-based user location and tracking system. In *Proceedings IEEE INFOCOM 2000. Conference on Com-*

*puter Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, volume 2, pages 775–784 vol.2, 2000.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] Flaviu Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3(3):146–158, September 1989.

[8] 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.

[9] 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.

[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] 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.

[13] 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.

[14] 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.

[15] Hwymeers. An example factor graph, November 2008.

[16] Kauai Labs Inc. Video Processing Latency Correction Algorithm, 2017.

[17] Itseez. OpenCV, Optical Flow, 2017.

[18] Marcoe Keith. LIDAR an Introduction and Overview, 2007.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] Muñoz-Salinas, Rafael, Marín-Jiménez, Manuel, Yeguas-Bolivar, Enrique, and Medina-Carnicer, Rafael. Mapping and Localization from Planar Markers. *Pattern Recognition*, 2016.

[30] NASA. Kalman Filter Integration of Modern Guidance and Navigation T1c Systems, 1999.

[31] Peter O'Donovan. Optical Flow: Techniques and Application, April 2005.

[32] Pozyx. Pozyx - centimeter positioning for Arduino, 2017.

[33] Richard Hartley and Andrew Zisserman. Multiple View Geometry in Computer Vision, 2003. p312.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] Min Sun. Optical Flow, 2008.

[39] 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.

[40] 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.

[41] 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.

[42] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[43] Lidar UK. How does LiDAR Work?, 2017.

[44] 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.

[45] 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.

[46] A. Ward, A. Jones, and A. Hopper. A new location technique for the active office. *IEEE Personal Communications*, 4(5):42–47, October 1997.

[47] 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.

[48] 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.

[49] Zebra. Dart Ultra Wideband UWB Technology | Zebra, 2017.