AutoUmp

Timothy Adams, CSE; Matthew Barnes, EE; Jason Camiel, EE; Justin Marple, CSE

Abstract — **Determining strikes and balls accurately and efficiently is a core aspect of baseball, but a way to do so has not yet emerged for little-league and pick-up games. Umpires are inaccurate or biased, and current technology solutions used in the MLB are prohibitively expensive. We introduce AutoUmp, a self-contained pitch determination solution installed in the home plate itself. AutoUmp uses two cameras, XMOS processors, and real-time image processing algorithms to detect strikes and balls. Currently, we have demonstrated our core image processing algorithm in MATLAB, and can detect motion in real-time on our prototype. Future work will focus on implementing the algorithm in C and developing a hardware prototype capable of meeting our specifications.**

I. Introduction

DETERMINING strikes and balls accurately and efficiently is a core aspect of the game of baseball, yet doing so without a professional umpire is difficult and inaccurate. In order to do so, an umpire must decide if a pitch has passed through the strike zone, a variable volume that consists of the space above home plate and between the batter's knees and halfway up their torso. Even in little league games, pitches can easily be thrown in the region of 70 miles per hour (mph), reaching the home plate in 0.65 seconds [1]. This difficulty directly correlates to incorrect calls, which can lead to significant frustration from players, coaches, and spectators alike, especially in a close game. In the MLB, where salaries for umpires range between $120,000 and $350,000 [2], 15% of strikes are called as balls and 13.2% of balls are called as strikes [3]. Extrapolating these statistics to little league games that are overseen by amateur umpires, one can imagine the extent to which games can be affected by incorrect calls.

Though the challenges associated with accurate pitch calling are widely apparent, little has been done to seek to resolve the problem. The MLB has adopted PITCHf/x technology to augment calls made by umpires, which is capable of tracking the entire trajectory of each pitch [4]. However, the technology is extremely expensive and only installed in MLB baseball stations. It consists of two cameras installed in the stands above home plate and first base, which capture approximately 20 images of the pitch during its flight and feed this information to a high-speed computer to determine the pitch's 3-D trajectory through space [5]. The expensive and time-consuming installation costs prohibit similar technology from being widely adopted anywhere except the MLB level. Other solutions frequently employed, such as having a catcher or a coach act as umpire, do little-to-nothing to solve the inherent problem and can introduce bias into the calls.

Therefore, there still remains a need for a portable, unbiased, and accurate pitch detection solution that can be used effectively by little-league teams and pick-up games. This is the niche that AutoUmp seeks to fill.

Requirement	Specification
Meet or exceed accuracy of human umpire	Determine strikes with 15% error or less when pitch is within one ball length of edge of strike zone
Accurately detect pitches for the users up to heights of 6' 6"	Detect pitches up to 2 meters high and 0.75 meters to either side of plate
Detect pitches at speeds up to 70mph	Cameras and image processing algorithm must run at 60fps
See pitch in at least 2 frames	At 60fps, field of view must exceed 93 degrees
Real-time use	Provide pitch determination within 3 seconds of pitch passing plate
Little-to-no setup required	Self-contained system
Robust system	Perform consistently in cloudy and sunny conditions; system can withstand impacts of normal play

Table 1. List of System Requirements and Specifications.

AutoUmp will consist of a self-contained, battery operated system stored in the home plate itself, with results sent to the user via LEDs in the plate and an Android app connected via Bluetooth. The user of the app will be able to input the batter height for an accurate determination of the strike zone, see the calls the system makes, and edit any calls if needed. Inputting the height of the batter provides a quick, easy way for the system to determine the height of the top and bottom of the strike zone by using data to find an approximation of knee and mid-torso height of people.

Through our team's own experience in baseball and market research, along with the feedback from other college-level baseball players, we have determined the requirements and specifications needed for effective implementation,

T. Adams, author from Arlington, MA (email: tbadams@umass.edu).
M. Barnes, author from Southborough, MA (email: mebarnes@umass.edu).
J. Camiel, author from Framingham, MA (email: jcamiel@umass.edu).
J. Marple, author from Pepperell, MA (email: jmarple@umass.edu).

outlined in Table 1. As our target market is youth to young adult baseball and wiffleball, the heights of players are far below our 6 feet 6 inches requirement, and it is unlikely that any ball will be thrown higher than that. It is fairly routine to see balls in the 50-70mph range in youth baseball; a frame rate of 60 frames per second (fps) enables us to effectively capture pitches at this speed without requiring a fisheye lens, significantly increasing the ease of implementing an accurate algorithm. This system is designed to substitute for an umpire, so the pitch call needs to be made quickly enough that play is not significantly slowed in comparison with a human umpire. Finally, it is a core part of baseball to step on the home plate, so the system must be robust to withstand such impacts.

II. Overview

To create a self-contained system like the one described, we will embed two cameras into the home plate, protecting them with a polycarbonate material. We monitor each camera for a ball flying overhead and, if seen, determine the point at which it crosses the strike zone. For our purposes, we model the strike zone as a plane, rather than a volume. Using just one camera does not provide enough information for our purposes, as each pixel location corresponds to a line in 2-D space where the ball might be. Using two cameras, however, allows us to find the intersection between these two lines and calculate both the x- and y-coordinates of the ball as it passes through the strike zone plane.

We considered several other sensing technologies, including radar and ultrasound. We found radar was extremely noisy and would require a prohibitively large antenna to achieve the wavelengths required to identify a baseball. Ultrasound seemed promising and afforded the potential of extremely cheap sensors, but also suffered from significant noise. Due to the maturity of optical sensors and the ability to easily detect moving objects via background subtraction, we opted to stay with image sensors.

Each camera must be capable of capturing data at 60 frames per second with a 320x240 resolution and a 95-degree field of view. These values enable us to see a ball in at least two frames, which is a requirement of our image processing algorithm while also reliably distinguishing a ball from stray noise in the image.

The hardware that interfaces with the cameras and runs the image processing algorithm must be small enough to fit inside the plate and fast enough to both read 4.6MB/sec of data from the cameras (320*240 pixels per frame, at 60 frames per second, where each pixel is one byte) and execute our image processing algorithm. We chose the XMOS 500Mhz [6] 16-core processor for this purpose. We had originally considered using an FPGA for the same purpose, but decided on the XMOS due to its ability to allow us to write all of our algorithms sequentially in C rather than explicitly in parallel for the FPGA, aiding greatly in reducing code complexity and testing. Figure 1 shows the relationships and protocols used between the different hardware blocks. The cameras will send image data to the XMOS processors. Each XMOS processor will perform sufficient image processing on the camera data to identify and locate a ball flying through the image (see Cores 1-4 in Fig. 2). To accomplish this the processors will perform background subtraction on consecutive images to identify objects in motion (Core 2). The result will be denoised (Core 3) and a flood fill algorithm will detect the ball (Core 4). The ball’s location from each camera will then be sent to the master processor, which will combine the information to determine whether the pitch crossed the strike-zone The results will be displayed via LEDs on the plate itself, and sent to the user via Bluetooth to an Android app

Serving as the primary point of interaction with the user, our app will display the current pitch count and general game information such as the score and inning number. Although both Android and iOS are equally acceptable options, we opted to develop and Android app due to a majority of the team having access to one and overall larger market share.

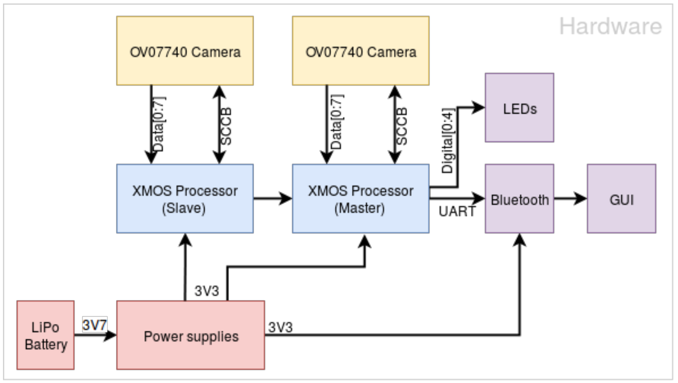

Fig. 1. Hardware Block Diagram.

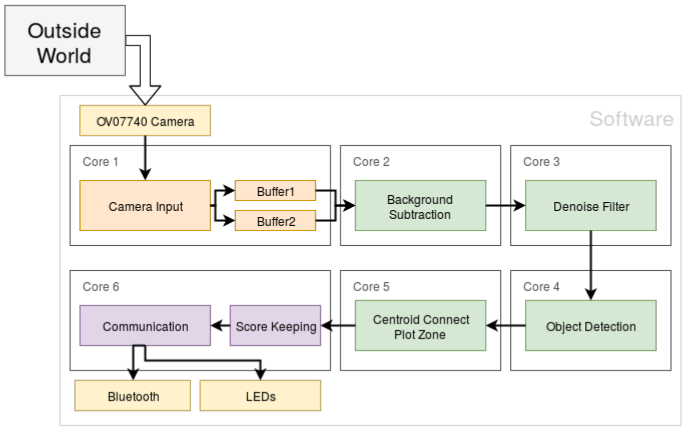

Fig. 2. Software Block Diagram.

A. *Cameras*

Capturing and utilizing data from two cameras is in many ways the central component of our project. Choosing the correct camera sensors and lenses requires balancing a number of camera properties as well as cost. In order for our algorithm to work correctly, we need to capture the ball in at least two frames and be able to see the ball up to a height of two meters. Increasing the framerate at which we capture will increase the number of frames where we will see the ball, and increasing the resolution increases our capability to see the ball and distinguish it from other noise. However, higher resolution and framerate will result in higher computation requirements, as our image-processing algorithm will need to operate on a significantly greater number of pixels. Another property that can be manipulated is the field of view, which describes the angle of the image cone that the camera can see. Increasing the field of view significantly can result in distortion of the image, which can affect the accuracy of our calculations. The field of view is entirely determined by the lenses we choose.

Most available camera sensors support up to 60fps. Using this value as a reference, we calculated that a field of view of 95-110 degrees will allow us to meet our >2 frames requirement (see Table 1). Furthermore, a resolution of 320x240 will render the ball with 25-30 pixels, even 2 meters above the plate, providing enough information to identify it distinctly as a ball.

We chose to model the strike zone as a plane in order to only use two cameras and to allow for the algorithm to run fast enough in order to process images faster than they are being recorded. Therefore it is theoretically possible for a ball to "hook" around the strike zone plane and intersect the volume above the home plate. In practice this is highly unlikely and most umpires would still call it a ball despite it crossing within the volume of the plate.

B. *XMOS Processor and Camera Communication*

In order to read our raw data from our camera sensors fast enough, we require an efficient hardware interface. This was one of the reasons why we chose the XMOS 500Mhz 16-core processor. This processor functions similar to an FPGA, in that the hardware interface can be written in software, but unlike an FPGA, the code can be written sequentially in C. In our case, the hardware interface will need to be a databus to the Omnivision cameras. Each camera has a HREF, VSYNC, PCLK, 8 data lines, and a SCCB line. HREF goes high every time a row starts, VSYNC goes high at the beginning of each image and PCLK goes high every time a pixel is set. SCCB is a protocol similar to I2C, which allows settings in the camera sensor to be set.

On the XMOS processor, each of these lines is hooked up to a 1-bit port, besides the data lines which are connected to an 8-bit port. A "port" in XMOS terms is a series of digital inputs that can be sampled at once. For instance, an 8-bit port can sample 8 pins on a single clock cycle, making it ideal for wide buses. Each port can also be sampled and buffered automatically using an input clock. So when sampling the pixel clock (PCLK) that can be used to tell the hardware when to sample the data lines. This makes efficient use of the hardware and only requires software to save the buffered samples to RAM when the buffer fills up.

C. *Image Processing*

The image processing block is the main block of the project, and is represented by Fig. 2. This block is run entirely on the two XMOS microprocessors that are each connected to a camera, and will be programmed in XC, a variant of C that is specific to XMOS. Each camera will be recording at 60 frames per second. Because a ball may fly through the field of view of the cameras at any time, each frame must be processed to determine if a ball has passed through the camera. Furthermore, each frame must be processed before the next frame is read, resulting in a 16.67ms time window. If a ball is found in the frame, then the ball's location is passed to the "Centroid Connect" and "Plot Zone" functions to determine where in 3-D space the ball passed through plane of the strike zone. Here, our time-constraint is much more lax, as the user can accept up to a 2-3 second delay in the pitch determination.

To meet our 1/60ms time requirement for ball detection, our algorithm is highly parallelized, where each method of our algorithm runs on a separate core of our XMOS microprocessor. To start, we perform background subtraction and thresholding on the current frame and the previous frame, creating a new image that has white pixels only where an object has moved since the last frame. This new image is passed through a denoising filter to eliminate stray noise or camera malfunctions that may be incorrectly labeled as objects.

The denoised image is then passed to an object detection function, which has two main components. First, a flood-fill algorithm identifies connected regions of white pixels, giving each a unique ID and summary information of each in a table. Second, every object ID that is connected to the edge of the image is removed. We perform this second step because the only object we care about – the ball – is also the only one that will be *in flight.* As our cameras are pointing straight up, any moving object not in flight must be connected to the edge of the image. This second step then allows us to identify the ball rather than falsely detecting a player's hand, the bat, etc.

Because of the nature of the background subtraction, the images we will operate on will have two "balls" in them – each representing the location of the ball in a different frame. We can use this fact to our advantage to greatly simplify our calculations by connecting the centers of these two balls and finding the point at which they cross the middle of the image. There will be at least one background subtracted image, for each camera and each pitch, that will contain two baseballs. By connecting the center of each of these baseball "objects" we can interpolate and find the intersection of this line with line of pixels that represents the area (plane) directly over the cameras. Rather than attempt to calculate the 3-D location of a ball and seeing if it passes through the volume of the strike zone, we model the strike zone as a plane, and represent the line of pixels that make up the middle of the image as that plane. This allows us to convert a 3-D problem into a 2-D problem. The last step of our algorithm takes the point in each image where the ball passes through this plane, and uses simple geometry and the properties of the camera such as field of view to calculate the x- and y-location of the ball in the plane of the strike zone (See Fig. 4a and Fig. 4b). From here, it is a simple matter to match the location of the ball with the strike zone computed from the batter's height.

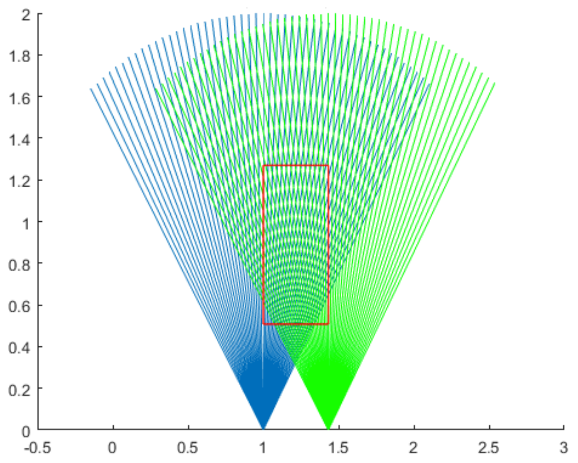

Fig. 4a. Using pixel location to determine ball location. Each line represents the possible location of an object given its location in a particular pixel of the image. The field of view affects how far apart these lines are spread. This simplified drawing uses only 50 pixels – our actual cameras use 240, allowing for a more precise calculation.

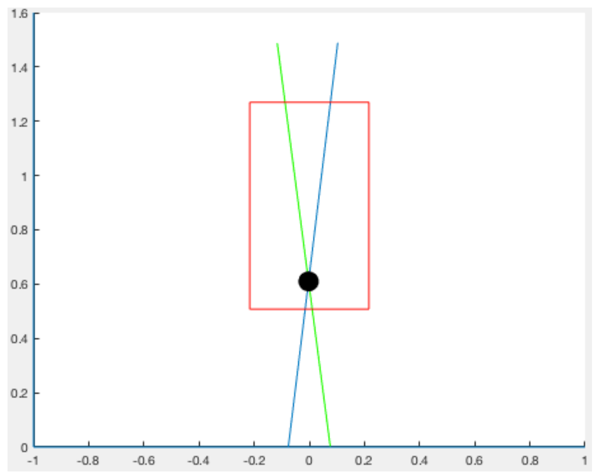

Fig. 4b. Once the center of the ball has been found in each image, we can use the pixel location to triangulate the ball's flight through the plane of the strike zone.

As the majority of this block is written in C, it utilizes all of the work conducted in those classes which have required us to program in C. Furthermore, team member Tim Adams took Image Processing this past semester, and has incorporated what he has learned into this section as well. Moving forward, however, we may need to study more image processing techniques to account for lens distortion or other errors that may affect the accuracy of our calculations.

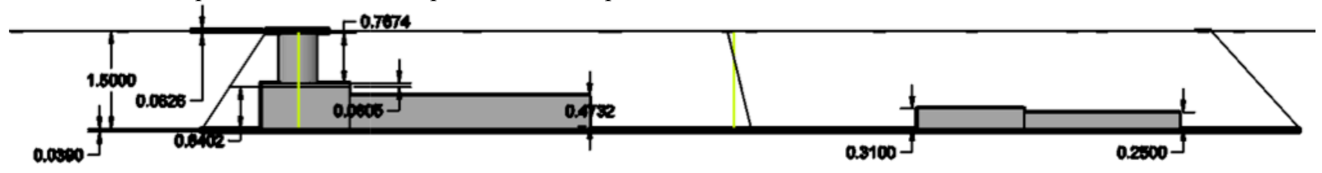

Fig. 5. Side view of enclosure and components inside. Design leaves significant amount of empty space if more battery storage is required.

We have already designed multiple experiments to test this block. Our most efficient and accurate experiment proceeds as follows: identical cameras are placed at a fixed distance apart on the ground and are pointed directly up. A strike zone and axis are drawn on a board or wall. One person throws multiple baseballs at the wall or board so that the pitches travel over the cameras, and a separate person marks the location that the ball strikes the board or wall. These positions are taken to be the points at which the ball crossed the strike zone plane at the front of the plate. When these videos are run through our MATLAB algorithm, the measured positions are compared to the results of the image-processing block. By repeating this type of experiment several times, we can develop a suite of tests that test all of our most difficult edge cases, as well as more typical pitches that will be seen in practice.

D. Enclosure

For the home plate system to perform effectively in a real-world game environment, we require a strong enclosure that will protect our electronics from damage. Normal game play will see the plate being stepped on or slid into by a player and hit by a bat.

Using AutoCAD, we were able to draft plans for placement of the hardware. The standard 1.5-inch depth plate provides plenty of space for extra battery storage or any unforeseen corrections to our hardware. Fig. 4 shows we could stack 3 batteries on top of each other and effectively triple our power storage, while still comfortably fitting within our space constraints.

In addition to fitting our hardware, our enclosure must provide suitable support to withstand normal gameplay. Fig. 5 shows the top view of our hardware placement, and provides insight into how we can support the enclosure. Using continuous cross braces will hold the integrity of the plate while minimizing any flex that could damage the components held within. Areas of white free space show good places for bracing, and as can be seen from Fig. 5, there is a significant amount of free space available.

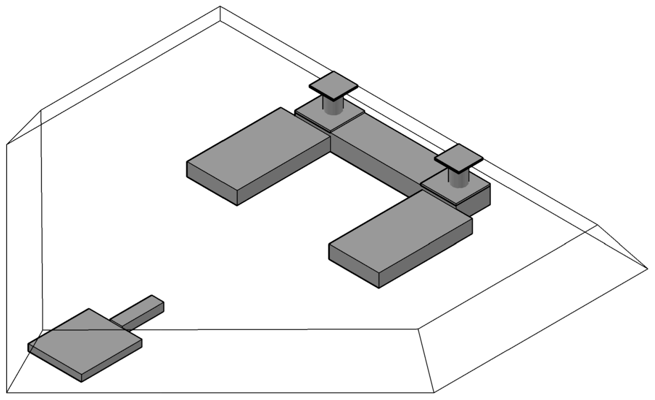

Fig 4. Top view of enclosure and components inside.

The backside of the plate will be sealed with one continuous piece of material, most likely sheet metal to help dissipate heat and provide structural support. This will complete the prism, strengthening the entire structure. In order to enable the camera lens to see the pitch, the top of our plate must be compromised. Thin sheets of a polycarbonate will weatherproof and shock proof the lenses while keeping optical distortion at a minimum. The polycarbonate material will be layered both under the plane of the top of the plate and above the plane of the top of the plate. This will allow for backup protection in case one fails. To ensure waterproofing around the entire enclosure, all joints will be caulked or sealed with rubber cement.

E. App

The app is the main interaction point between our system and the user. It will be the only place the user can interrupt and change calls, and in tandem with LEDs on the plate, will indicate the called pitches provided by the system. Developed for Android using Android Studio [7], the app will allow users to start a new game and pair with their plate via Bluetooth. After doing so, the main game screen appears, where the user can view and update the current pitch count, score, and inning (see Fig. 6). All labels double as buttons which increment their respective values, rolling over to 0 if they exceed the maximum possible value (i.e. 3 for outs or strikes and 4 for balls). An edit screen will also be available to edit score or inning number if these values are accidently incremented.

We require two-way communication between the app and the home plate for complete functionality. The home plate will send pitch information to the app, along with its current reading of general game information such as current inning, number of outs, and score. The app will send the home plate any updated information the user might provide about general game information or the current pitch count, as well as the current batter's height.

So far, we have sketched the basic UI interface, but we still need to implement the core functionality and Bluetooth pairing. To accomplish this, we will build on our experience using Java in our previous courses. Namely, we need to learn how to interface our app with a Bluetooth device, and seamlessly update the pitch count based on new information from the paired device.

Our app can be tested independently of the rest of the system by manually inputting test cases via the console to our app. The test cases will be formatted in the same way as the data received via the Bluetooth interface from the app, and will allow us to verify that all user interface components work successfully. After communication between the microprocessors and app is established, the same test cases can be sent from the microprocessor.

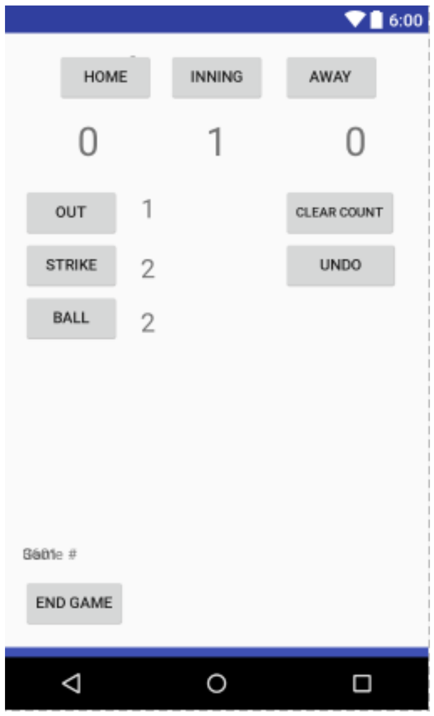

Fig 6. Main page of app. Each label also acts as a button that can increment its given number.

III. Project Management

Promised	Delivered?
Demonstration of image processing algorithm (MATLAB)	Yes: given two .MOV files, returns pitch determination
Demonstration of image collection	Yes: prototype system collects live video from two cameras and lights up sections 3x3 LED grid corresponding to location detected motion in image

Table 2. Promised MDR Deliverables.

Our team successfully accomplished both of our MDR deliverables as outlined in Table 2. As promised, we completed our core image processing algorithm in MATLAB, demonstrating that our basic method is sound and achievable. We recorded videos using our pre-described testing and then demonstrated that given two video files representing the different cameras, our MATLAB code could take, perform our entire algorithm and output the location of the pitch in 2D space where it crossed the plate. We also developed a prototype that not only can collect image data, but can process that data in real time to detect motion, displaying the first step of our overall image processing algorithm.

Moving forward towards CDR, we need to design and ship out a PCB for our cameras and microprocessors. Our current prototype is only capable of recording at 30 frames per second, and lacks the field of view required to meet our specifications, and so we have ordered an entire suite of new parts to build a fully functional prototype. In the first few weeks of the semester, we will combine these parts and our PCB to create a prototype. Once our hardware is complete, we can begin building our enclosure.

Concurrent to this work, we can develop our software components, both for our core image processing algorithm and the Android app. The image processing algorithm will first be ported over to C, and then, once it has gone through an initial round of testing, will be translated to XC, the variant of C that runs on our XMOS microprocessors. The app will be developed relatively independently of the rest of the system, establishing Bluetooth connectivity in Mid-February. The Gantt Chart (Fig, 7) provides an overview of our plan.

Our team dynamics have been quite good, demonstrated by our ability to continue to work together and remain in good spirits despite several setbacks this past semester. Team management was made more difficult throughout the majority of the semester, as Justin was in New Zealand studying abroad until mid-November. As a result, we met weekly with the four of us at a time when we were all available, and then the three of us in Amherst met weekly with our advisor Professor Wolf, recording and updating Justin after each meeting. Most of our communication has occurred over Facebook Messenger, which has allowed us to both message and conference call with the entire team as Justin has been abroad.

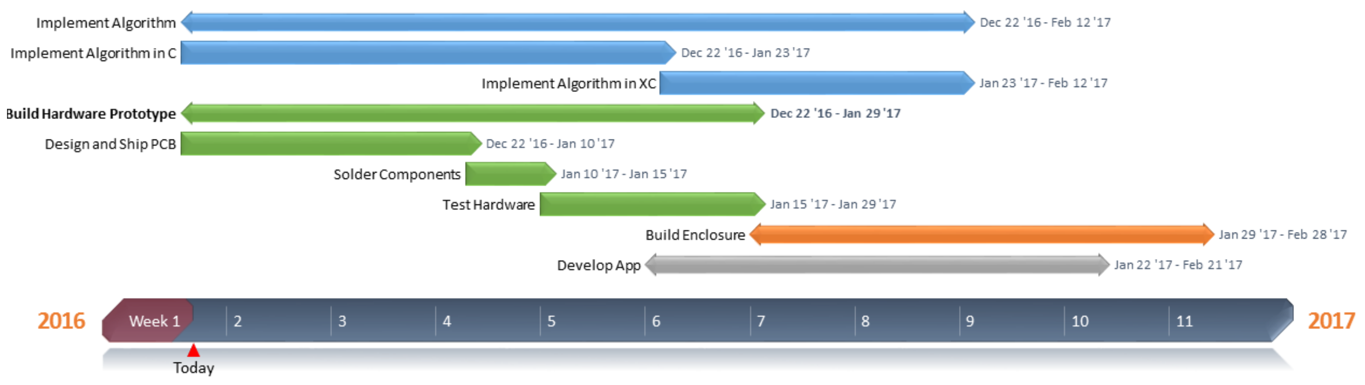

Fig. 7. Gantt Chart of CDR Deliverables.

Throughout the fall semester, much of the work we have done has been done collaboratively, with team members often overlapping in responsibilities and taking turns driving the project forward at different points in the semester. Much of the work this semester involved determining the specifications necessary for our project, which was truly a joint effort. Justin has taken the lead in interfacing with the cameras and designing the PCB, and will continue to lead in those capacities. Matt and Tim have collaborated extensively on developing the overall algorithm in MATLAB. Moving forward, Tim will take the lead in developing the algorithm in C, and work with Justin to translate it to XC. Jason and Matt have worked together to develop a large number of test cases that will prove useful in testing our algorithm. Jason has designed the enclosure in AutoCAD and led the way in tracking down the surprisingly difficult-to-find camera lenses and other parts that are required for us to meet specification.

IV. Conclusion

Much work has been completed at this stage of our project. At this point, we have successfully demonstrated our image processing algorithm, established image collection and real time image processing on a prototype, and ordered all parts necessary to create our final prototype. We have collected a variety of test data that will allow us to test our algorithm, built a user interface for our app, and begun to design an enclosure that will protect our system from normal game-play.

Moving forward, we will design and build our PCB and construct our final prototype. Concurrently, we will implement and test our algorithm in C, begin to build an enclosure, and develop our app. We anticipate that the most significant difficulties will come from the hardware design of our prototype as our PCB is fairly complex and may require a couple of tries to be completely successful. We will buy ourselves as much time as possible by developing and testing the algorithm as much as possible in C so that our software and logic is essentially completed by the time our hardware is ready. We can also learn how to pair our app to Bluetooth and finish the app entirely independent of the prototype being completed.

Acknowledgments

Many thanks go to our advisor, Professor Tilman Wolf, who has provided strong guidance and support throughout our project.

References

[1] "Pitching Speeds," *Steven Ellis Pitching Tips* [Online]. Available at: http://www.thecompletepitcher.com/pitching_speeds.htm. [Accessed: Dec-2016].
[2] "Much required to become MLB Umpire", *MLB.com* [Online]. Available at: http://m.mlb.com/news/article/2173765//. [Accessed: Dec-2016].
[3] "How well do umpires call balls and strikes?" *Beyond the Box Score* [Online]. Available at: http://www.beyondtheboxscore.com/2014/1/27/5341676/how-well-do-umpires-call-balls-and-strikes [Accessed: Dec-2016].
[4] "PITCHF/X". *Sportvision* [Online]. Available at: http://www.beyondtheboxscore.com/2014/1/27/5341676/how-well-do-umpires-call-balls-and-strikes [Accessed: Dec-2016].
[5] "What the Heck is PITCHf/x?", *The Hardball Times Baseball Annual 2010* [Online]. Available at: http://baseball.physics.illinois.edu/FastPFXGuide.pdf. [Accesesed: Dec-2016].
[6] "XUF216-512-TQ128 Datasheet", *XMOS*. Available at: https://www.xmos.com/download/private/XUF216-512-TQ128-Datasheet(1.11).pdf [Accessed: Feb-2017]
[7] "Android Studio", *Android*. [Online] Available at: https://developer.android.com/studio/index.html. [Accessed: Dec-2016]