AutoUmp

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

Abstract—Determining strikes and balls accurately is a core aspect of baseball. However, umpires are inaccurate or biased, and current technology solutions used in the MLB are prohibitively expensive. AutoUmp is a self-contained pitch calling solution installed in the home plate itself. It uses optical sensors and real-time image processing algorithms to detect strikes and balls.

Index Terms—Stereo Cameras, Image Processing, Baseball, Embedded System

1 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 batters 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].

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 stadiums. 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 pitchs 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. 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.

AutoUmp consists of a self-contained, battery operated system stored in the home plate itself, with results sent to the user through an Android app connected to the plate via Bluetooth. The user of the app can 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 the batter.

TABLE 1
Specifications

Specification	Goal	Actual
Accuracy of professional ump	85% accuracy in pitch classification	83% accuracy in pitch classification (n = 100)
Useable for batters up to heights of 6'6"	Detect 70 in up, 30 in to each side	Detect 60 in up, 20 in to each side
Pitch speeds of 70 mph	Operate at 60 frames per second	Operate at 60 frames per second
Real time use	<2 second delay	<1 second delay
Battery life greater than length of game	3 hour battery life	20 hour battery life (2.2W operation)
Robust, self-standing system	Self-enclosed, withstand impacts of normal play	Self-enclosed, withstand impacts of normal play
Enable varying heights	Control strike zone via app	Control strike zone via app

Through our teams 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, outlined in Table 1. As our target market is youth to young adult baseball and wiffle ball, 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.

T. Adams, author from Arlington, MA (email: tbadams45@gmail.com).
M. Barnes, author from Southbourough, MA (email: mbz1918@gmail.com).
J. Camiel, author from Framingham, MA (email: jcamiel@umass.edu).
J. Marple, author from Pepperell, MA (email: jmredsoxfan1010@gmail.com).

2 OVERVIEW

To create a self-contained system like the one described, we embedded two cameras into the home plate, protecting them with sapphire watch crystals. We monitor each camera for a ball flying overhead and, if seen, determine the point at which it passes through 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 2-D vector of where the ball might be in the plane of the strike zone. Using two cameras, however, allows us to find the intersection between these two vectors 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 106-degree field of view in the direction of the pitch. 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 $4.6\frac{\text{MB}}{\text{sec}}$ of data from each camera and execute our image processing algorithm ($320 * 240\frac{\text{pixels}}{\text{frame}} * 60\frac{\text{frames}}{\text{sec}} * 1\frac{\text{byte}}{\text{pixel}} = 4.6\frac{\text{MB}}{\text{sec}}$). We chose the XMOS XUF216-512-TQ128 16-core processor for this purpose [6]. 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 in C rather than in Verilog for FPGA, aiding greatly in reducing code complexity and testing.

Fig. 1 outlines the functional blocks and interfaces connecting them at the hardware level. We use a single XMOS 16-core processor, split into 2 *tiles*, which act as miniature 8-core processors with their own dedicated memory with a highly optimized communication interface connecting them. Fig. 2 outlines the tasks of the different cores. The cameras each send image data to the camera master tile, which performs background subtraction on consecutive images to identify objects in motion. This data is then sent to one of 6 cores, which denoise and identify objects in the image using a flood fill algorithm. The ball's location from each camera is sent to an object tracker core, which determines if the ball has crossed the strike zone plane and, if so, calculates its exact position in the strike zone plane.

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 an Android app due to a majority of the team having access to one and overall larger market share.

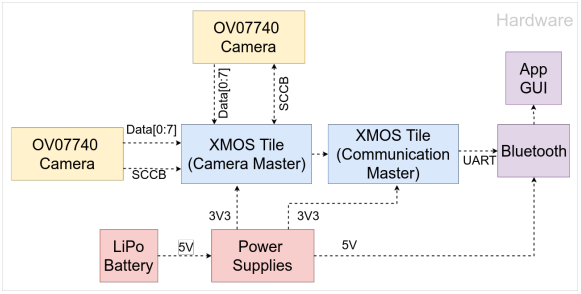

Fig. 1. Hardware Block Diagram

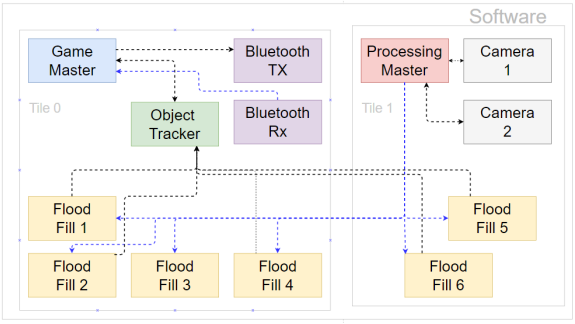

Fig. 2. Software Block Diagram

2.1 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 perform correctly and to meet specification, we need to capture the baseball in at least two frames and be able to see the baseball up to a height of 6 feet 6 inches. Increasing the frame rate at which we capture increases the number of frames where we will see the baseball, while increasing the resolution improves our ability to distinguish the baseball from other objects or noise. However, higher resolutions and frame rates 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 to that of a fish-eye lens (roughly between 130-180 degrees) can result in distortion of the image, which can affect the accuracy of our calculations.

Ultimately, we purchased the OV7740 image sensors from Omnivision, capable of 60fps at 320x240 resolution. These sensors use the SCCB interface to communicate with the processor, developed by Omnivision [7]. The custom PCB we designed around this chip is seen in Fig. 3. In addition, we used the JSD5011 lenses from Shenzhen JSD Optoelectronics Company, due to their 110 degree horizontal field of view and 80 degree vertical field of view.

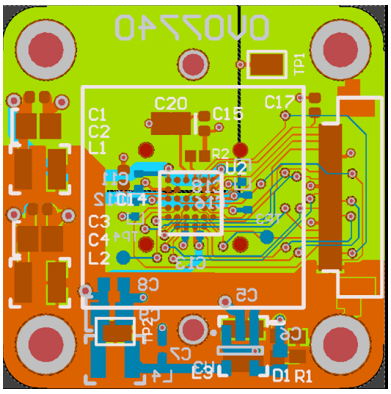

Fig. 3. Top layer of custom designed camera PCB

2.1.1 Frame Rate and Field of View

Our first step is field of view requirements for our lenses. There are two different field of views we care about: in the direction of the pitch, and perpendicular to the pitch. The first is used to ensure that the pitch is captured in at least two frames; the second is used to ensure that the entire strike and the nearby surrounding area zone is captured by both cameras. We examine each in turn.

Our field of view in the direction of the pitch is related both to the speed of the pitch and to our frame rate. Because we require two frames in which the pitch appears to perform our image processing algorithm, the ball cannot travel more than half of the image in one frame. At the same velocity, the ball travels "faster" across the image when its height is lower, as the amount of distance in real-world space covered by each pixel is less. We must therefore then set a low height at which to perform our calculations, in order to meet our worst case. For a 48 inch tall batter, the bottom of the strike zone is ˜15.6 inches. We set our height of the ball to be 14.6 inches, then, to be slightly lower than this.

The distance d_s at a given height h seen by a lens with a field of view of f degrees is

$$d_s = 2 * h * tan(\frac{radians(f)}{2}) \quad (1)$$

The distance d_t a ball travels during one frame is

$$d_t = \frac{speed}{fps} \quad (2)$$

where $speed$ is expressed in either $\frac{m}{sec}$ or $\frac{in}{sec}$, and fps is the frames per second of the image sensor. To meet our requirement, then, $d_s/d_t \geq 2$ must hold true at when $h = 14.6$ in. Solving for fps, we obtained a minimum field of view of 106 degrees as necessary in the direction of the pitch. Our final system had a field of view of 110 degrees in this direction.

In the plane of the strike zone, we required that both cameras see the entire strike zone and the inches surrounding the strike zone where a call may be disputed. Visually inspecting figures such as Fig. 7, which show the area seen by a camera with a given field of view, we determined that a field of view of 80 degrees would allow us to meet our requirements. In addition, we opted to tilt the cameras 15 degrees inward to make the most of the entire field of view available to us. This proved to be advantageous – our final image processing algorithm uses only the middle $3/4$ths of the image, restricting our field of view from our lens' spec of 80 degrees to 60 degrees. Fig. 7 shows the final field of views used by our algorithm.

2.1.2 Resolution

Image resolution was another important factor with a variety of trade offs. Increasing image resolution increases the ease of detecting a ball and distinguishing it from random noise, as well as increasing the accuracy of our calculations, especially at higher heights. At the same time, higher resolutions results in significantly more computational time required, as any doubling of the sides of an image results in a four-fold increase in pixels. Given our need to operate all of our object detection and tracking at 60fps, we required an image resolution as low as possible.

Our limiting factor, then, was the size of a ball at our maximum height of 6 feet 6 inches. Visually inspecting MATLAB plots of the kind see in Fig. 4 showed us that a resolution 320x240 pixels (QVGA) would still allow us to see the ball in 25-30 pixels, providing enough information to identify it distinctly as a ball.

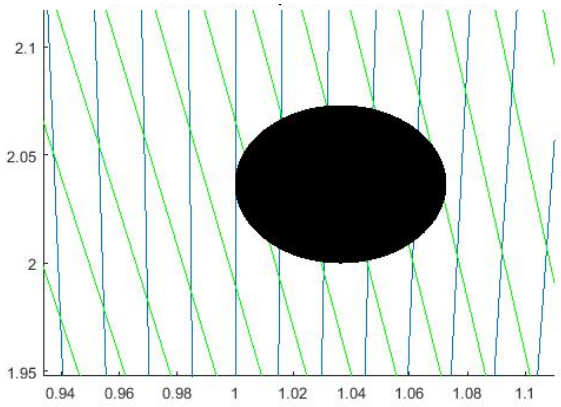

Fig. 4. Intersection of ball with various pixel rays. This demonstrates a worst case: a field of view of 105 degrees with a resolution of 240 pixels, with the ball 6 feet 6 inches high. Even so, we see the ball with 5 pixels from each camera, for a total of approximately 25 pixels.

2.2 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 XUF216-512-TQ128 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 needed to be a databus to the Omnivision cameras. Each camera has a HREF, VSYNC, PCLK, 8 data lines, and a SCCB interface. 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.

Due to the need to implement this interface directly, the need for a small form factor in the plate, and the lack of available plug-and-play options for the XUF216-512-TQ128, we opted to design our own PCB for this chip, the top layer of which can be seen in Fig. 5.

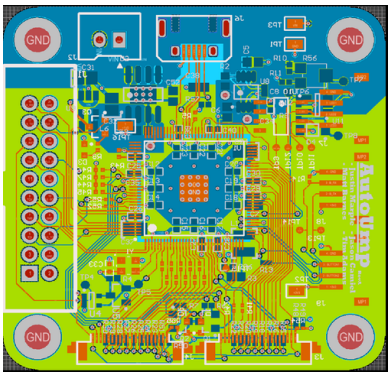

Fig. 5. Top layer of custom designed XMOS Processor PCB

2.3 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 XMOS microprocessors that is connected to both cameras and is programmed in XC, a variant of C that is specific to XMOS [8]. Each camera captures data at 60 frames per second. Because a pitch may fly through the field of view of the cameras at any time, each frame must be processed to determine if a pitch has passed through the camera. Furthermore, each frame must be processed before the next frame is read, resulting in a 16.67ms ($\frac{1}{60}$ ms) time window.

Our image processing algorithm is pipelined, with each section working on a different frame or set of frames. We begin with performing background subtraction on the raw image data to detect motion, a process where the current frame is subtracted from the previous frame. The resulting image removes the background and sets objects in motion as white pixels. In addition, this image appears to have two different balls in it, but which really represent the ball at the two different time points the frames were captured.

The next steps, denoise and object detection (also known as "flood fill", see 2), are parallelized across 6 cores, as they are by far the most computationally expensive step. In the denoising step, pixels are set to black if less than 3 of their 4-connected neighbors are white. Object detection then begins by finding connected sets of white pixels. The output is an array of objects, each modeled as rectangles.

These arrays are passed to an object tracker core, which unites the information from all 6 cores to track a pitch. When the ball passes the middle of the screen, the pixel at which its trajectory along the middle column is calculated and a flag is set for this camera. This pixel represents the vector in the strike zone plane where the ball may be. When both flags are set, the information from both cameras is combined and the pitch is calculated. The result is then sent to the app via Bluetooth.

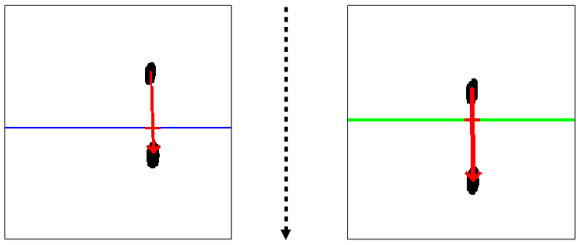

Fig. 6. Finding the intersection of the ball with the strike zone plate in each camera.

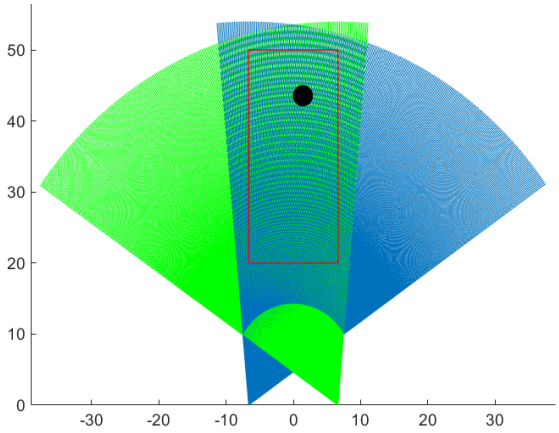

Fig. 7. Axes are in inches; the blue/green region indicates the field of views of the left/right cameras used by the algorithm to locate the pitch in the strike zone (shown in red).

2.3.1 *Meeting Real-Time Deadlines*

In order to meet our real-time deadlines, we had to optimize several steps of our code, and utilize the full processing capability offered us by the XMOS processor. The background subtraction step is implemented in assembly, so as to allow it to run fast enough to run on the same thread that collects the raw camera data. This freed up a core for other processing and minimized data transfer. The denoising step was optimized by including a look-up table that enabled us to examine 4 pixels at the same time.

Though these optimizations improved our performance, our most significant bottleneck occurred at the object detection stage. Our implementation of flood fill, a simple object detection algorithm, had a final worst-case run-time of 72ms, if the image was all white. To overcome this, we parallelized this stage across 6 cores (3 for each camera). Each core receives a frame, and operates the full flood fill algorithm on it. However, rather than only having 16.67ms to perform the algorithm, it has three times as much time, as the next two frames are being handled by other cores. This allowed us to increase our time constraint from 16.67ms to 50ms. From here, we relied on the fact that the images being processed were background subtracted images, i.e. images

that showed change between one frame and the next. With this being the case, we judged it impossible for more than half of the image to be white in our use-cases, enabling us to meet our requirement.

2.3.2 Calculating the Location of The Pitch

In order to calculate the final location of the pitch, we take as inputs two pixel locations, one from each camera. These values refer to the intersection of the trajectory of the ball with the middle of the image, as seen in Fig. 6. They also refer to a vector in the 2-D strike zone plane, indicating where the ball might be. We can imagine these vectors as being two sides of a triangle, where the third side is the line connecting the cameras together. This is shown in Fig. 8.

In our system, we have the following as constants:

- The distance C between the cameras, equal to 13.25 in.
- The location of each camera, designated by $[x_l, y_l]$, $[x_r, y_r]$.
- The field of view fov of each camera, equal to 80 degrees.
- The resolution res of the camera in the plane of the strike zone, equal to 240 pixels.
- The offset angles $\gamma_L = \gamma_R = \frac{180-fov}{2} - 15 = 35$ degrees.

We can calculate the number of degrees each pixel covers in the field of view with

$$\delta = \frac{fov}{res} \tag{3}$$

Furthermore, we take as input a pixel location from both the left and right cameras, denoted as p_l and p_r. These allow us to calculate the angles θ and α, using

$$\theta = \gamma + (res - p_l) * \delta \tag{4}$$

$$\alpha = \gamma + res * \delta \tag{5}$$

From there, β is trivially found to be

$$\beta = 180 - \theta - \alpha \tag{6}$$

And the length L can be found by the sine rule

$$L = C * \frac{sin(\alpha)}{sin(\beta)} \tag{7}$$

We can then find the location of the ball $[x_b, y_b]$ by

$$x_b = x_l + L * cos(\theta) \tag{8}$$

$$y_b = y_l + L * sin(\theta) \tag{9}$$

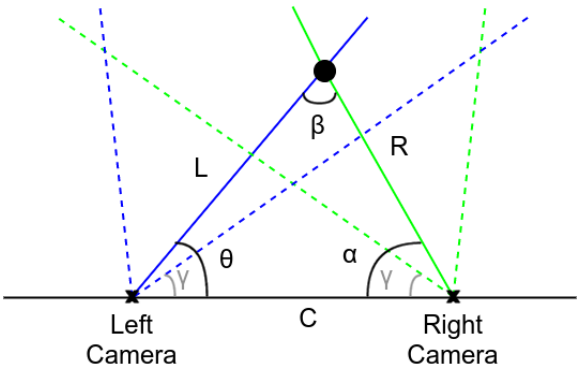

Fig. 8. The triangle used to calculate the ultimate pitch location. Solid lines indicate the vectors represented by the pixel the ball is found to be in. Dotted lines indicate the limits of the field of view for the left (blue) and right (green) cameras. Not drawn to scale – figure is for conceptual purposes only.

2.3.3 Limitations

Currently, our system is only capable of detecting a ball if there is no batter present. We were able to successfully implement a object tracking algorithm in C which was able to isolate a ball in the presence of batter and bat, and run this algorithm on our laptops. However, even on a laptop, this algorithm was slow, often taking a second or more to complete, and therefore orders of magnitude slower than necessary to meet our constraints. We therefore simplified our algorithm to only examine the middle $3/4$ths of an image and assume only a ball is present in this portion of the image. This allowed us to meet our real-time constraints - however, this also reduced our effective field of view in both dimensions, and decreased our maximum detectable speed to 52.5mph.

Though a more efficient implementation of the advanced object tracking algorithm could be implemented, we believe the primary constraint here is processing power, not coding ability. In order for such a system to work in the presence of a batter, more processing power would be required.

2.4 Enclosure

To perform effectively in a real world environment, the system requires an enclosure (Fig. 9) that will protect the electronics during normal gameplay, including being stepped on, slid into, or hit with a bat. Rather than attempt to build such a enclosure from scratch, we opted to adapt a real home plate to our purposes. Two holes were cut into the top of the plate to allow the camera to see through, while embedded sapphire watch crystals protect the lenses from shock, dirt, and water.

The sapphire crystals were selected due to their ranking on the Mohs hardness scale (9), which exceeds that of quartz (7), a material typically found in the dirt of a baseball diamond. The underside of the home plate was hollowed out to provide space for the cameras, processor, and the battery. 3D printed mounts raise the cameras to just beneath the crystals and are angled 15 degrees inward to allow each camera to see the entire strike zone.

The entire system is secured to an aluminum backing, which fits snugly into the plate. The aluminum backing is

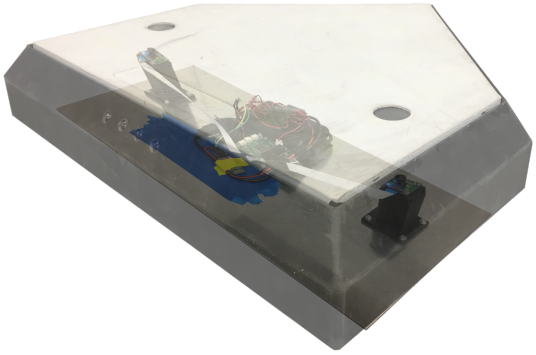

Fig. 9. The enclosure. The plate has been hollowed out to allow for the system to fit inside.

further secured by Velcro straps to ensure system stability throughout the game. These straps allow the backing to be removed to allow battery charging.

2.5 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 will indicate the called pitches provided by the system. Developed for Android, 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 (Fig. 10). 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). The user can also view the most recent pitch on a separate screen (Fig. 11).

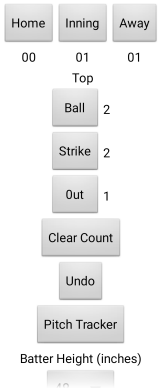

Fig. 10. The user can see a summary of the game and can override the system's call through the app.

We require two-way communication between the app and the home plate for complete functionality. The home plate maintains the complete game state, to allow a user to disconnect with one device and reconnect with another if needed. When a pitch is determined, the plate will update the game state and send this information over Bluetooth to the app, which will update its view with that state. Should the user decide to undo a call, increment a score, change the batter's height, etc., a command is sent back to the home plate, which maintains all the necessary logic to implement the command. In particular, a stack is maintained of the last 100 game states, so that the user can undo the last 100 actions.

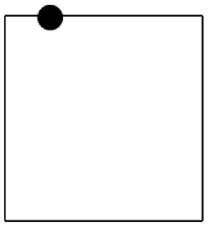

Fig. 11. The user can view the most recent pitch's location on a separate screen.

3 Results

Fig. 12 shows the results of 100 pitches. The location of the points indicates the actual location of the ball, which was wrapped in Velcro and thrown at a Velcro wall to allow accurate measuring. The color indicates whether the pitch was classified correctly as a ball or strike. With a couple of exceptions, our only errors stemmed around the edge of the strike zone itself, as would be expected. Errors could arise from a variety of factors. A small pixel error could impact the ultimate calculations, whether from (a) determining where the pitch crosses the middle of the image for either camera, (b) rounding errors, (c) modeling the pitch as a rectangle, or (d) plotting the trajectory of the pitch. In addition, our system is highly sensitive to objects moving inside the image that are not the ball. While our algorithm only considers objects inside the middle $3/4a$ths of an image, it is possible that one of our team members moving inside the image during testing could have resulted in some of the error seen.

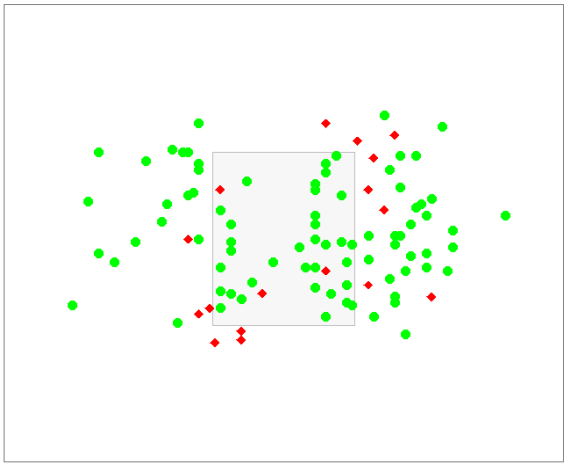

Fig. 12. Result of 100 pitches. Green means the correct call was made; red indicates incorrect call.

TABLE 2
Actual vs. Calculated Pitches

	Actual Strike	Actual Ball
AutoUmp Calculated Strike	34	14
AutoUmp Calculated Ball	3	39

Table 2 presents another view of the data, showing more clearly that our system experiences significantly more false strikes than false balls. When averaged out from an absolute perspective, our system had an overall accuracy of 83%, approximating that of a professional umpire. A comparison from such an absolute perspective is shown in Table 3. Breaking down our results further from Table 2, we see that for our system, 27% of balls are called as strikes and 8% of strikes are called as balls, as opposed to the 13.2% and 15% seen by professional umpires [3].

TABLE 3
AutoUmp vs. Competition

	Little League Umpire	MLB Umpire	Pitch FX (MLB)	Our System (AutoUmp)
Accuracy	about 60%	85%	99.9%	83%
Cost	$	$$$	$$$$	$

4 Project Management

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 year. Team management was made more difficult throughout the majority of the fall 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 was abroad.

Throughout the year, 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. However, there were one or two areas where each member contributed significantly in the spring semester. Justin was in charge of designing and soldering our PCBs, as well as implementing the SCCB interface and UART systems necessary to have each component communicate with one another. Matt was responsible for the enclosure, and worked out the final algorithm used to find the actual (x,y) position of the pitch using the vector information from both cameras. Tim was team manager, and responsible for implementing the image processing algorithms and optimizing these algorithms to meet our real-time deadlines – Justin also helped significantly in this latter area. Jason was responsible for developing the app, all of the ordering for our system, and building the Velcro wall we used to demo our system.

TABLE 4
Development Costs

Item	Cost ($)
Initial Prototype	380.26
PCBs and PCB Parts	478.28
Enclosure	198.87
Misc.	185.69
Total Cost	**1243.20**

TABLE 5
Production Costs

Item	Qty.	Unit Cost ($, Per 1000)	Total ($)
Optical Lens	2	1.50	3.00
10000mAh Battery	1	7.00	7.00
Home Plate	1	61.99	61.99
HC-05 Bluetooth Module	1	2.60	2.60
Sapphire Crystal	2	5.00	10.00
OV7440 Image Sensor	2	2.11	4.22
XMOS Processor	1	9.75	9.75
PCB Parts	1	11.47	11.47
XMOS PCB	1	3.99	3.99
Image Sensor PCB	2	3.86	7.72
Camera Mounts	2	4.07	8.14
Total Cost			**129.88**

5 Conclusion

AutoUmp is a working system that can detect balls and strikes. Its accuracy approaches that of a professional umpire for a fraction of the cost, and it is capable of detecting balls up to 52.5mph. Due to its long battery life, accuracy, adjustable batter height, and secure enclosure, it is highly suited for pitching practice. Limitations on processing power currently prevent the system from performing acceptably while a batter is present – however, including an extra processor in the system would likely provide enough computational power.

6 Acknowledgements

Our greatest thanks go to Professor Wolf for his advice, support, and his many jokes made with us and at us. Its been an honor to be your SDP team. Thanks also to Professors Anderson and Moritz for serving as our evaluators; Fran Caron for his help with our orders; Professor Hollot for his flexibility; Rick and Colby of the mechanical engineering machine shop for their asistance with the enclosure; and the ECE department for the extra funding.

References

[1] Steven Ellis Pitching Tips. *Pitching Speeds*. URL: http://www.thecompletepitcher.com/pitching_speeds.htm.

[2] MLB.com. *Much required to become MLB Umpire*. URL: http://m.mlb.com/news/article/2173765//.

[3] Beyond the Box Score. *How well do umpires call balls and strikes*. URL: http://www.beyondtheboxscore.com/2014/1/27/5341676/how-well-do-umpires-call-balls-and-strikes.

[4] Sportvision. *PITCHF/X*. URL: http://www.beyondtheboxscore.com/2014/1/27/5341676/how-well-do-umpires-call-balls-and-strikes.

[5] The Hardball Times Baseball Annual 2010. *What the Heck is PITCHf/x*. URL: http://baseball.physics.illinois.edu/FastPFXGuide.pdf.
[6] *XUF216-512-TQ128 Datasheet*. XMOS. Feb. 2017.
[7] *Serial Camera Control Bus Functional Specification*. Version 2.0. Omnivision. Mar. 2002.
[8] David May. *The XMOS XS1 Architecture*. XMOS. 2009.