Senior Theses, Projects, and Awards

Spring 2024

Design & Implementation of a More Precise Competition Timer in Ware Pool

Ben Freeman, ’24
Benjamin Wesley, ’24

Follow this and additional works at: https://works.swarthmore.edu/theses

Part of the Engineering Commons

Recommended Citation
Freeman, Ben, ’24 and Wesley, Benjamin, ’24, "Design & Implementation of a More Precise Competition Timer in Ware Pool" (2024). Senior Theses, Projects, and Awards. 925.
https://works.swarthmore.edu/theses/925

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License

Please note: the theses in this collection are undergraduate senior theses completed by senior undergraduate students who have received a bachelor's degree.

This work is brought to you for free by Swarthmore College Libraries' Works. It has been accepted for inclusion in Senior Theses, Projects, and Awards by an authorized administrator of Works. For more information, please contact myworks@swarthmore.edu.
ENGR 090 - Engineering Design - Design & Implementation of a More Precise Competition Timer in Ware Pool

5/10/2023

Ben Freeman, Benjamin Wesley

Advised by Professor Lynne Molter

Department of Engineering, Swarthmore College
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>3</td>
</tr>
<tr>
<td>Abstract</td>
<td>5</td>
</tr>
<tr>
<td>Introduction &amp; Motivation</td>
<td>6</td>
</tr>
<tr>
<td>Methods</td>
<td>9</td>
</tr>
<tr>
<td>Design Standards</td>
<td>15</td>
</tr>
<tr>
<td>Results</td>
<td>17</td>
</tr>
<tr>
<td>Discussion</td>
<td>23</td>
</tr>
<tr>
<td>Conclusion</td>
<td>26</td>
</tr>
<tr>
<td>Bibliography</td>
<td>27</td>
</tr>
<tr>
<td>Appendices</td>
<td>28</td>
</tr>
</tbody>
</table>
Acknowledgements

We would like to begin by thanking Professor Lynne Molter for advising us throughout the project. Her guidance and help with connecting us with Swarthmore College Facilities, coordinating weekly check-in meetings and thoughtful suggestions have been invaluable to the progress of this project.

Additionally, James “J” Johnson allowed us to use the machine shop and construct our camera frame. His guidance in teaching us how to use cutting equipment as well as lending us many unistrut parts to help cut costs in an efficient manner.

This project would also not have been possible without the help of the Swarthmore College Facilities team, specifically Senior Project Manager Tom Cochrane. Tom is well-known for helping seniors pursue their E90 projects and was a massive helping hand in allowing us to install our camera frame onto the ceiling of Ware Pool. Without his help in allowing us to use his team and their equipment, the quality of this project would not have been possible.

We would also like to thank the various members of the Swarthmore College swim team, most notably Sam Peterson and Maddie Adams-Miller ‘26. Having fellow swimmers with the same passion as us for a project so heavily involved around a pool alleviated some of the hurdles that might have otherwise been present in communicating our goals. For Adams-Miller and Peterson in particular, their adapted version of the Mathworks multi-object tracking example (Appendix A) provided a crucial foundation for the analysis of our data. This project would not have been significantly more challenging without their dedicated efforts and contributions.
Last but not least, we would like to thank Swarthmore College’s Head and Associate Swimming Coaches Karin Colby and Sam Davy for allowing us to use the pool space both during and after operating pool hours. Completing a project of this scale in an environment that many other students, faculty, and local patrons use can be incredibly tedious. However, they graciously gave us permission to use the space and complete our experiment.

This project was inevitably going to be a task more difficult than the two of us could handle alone. It involved the help of many Swarthmore College community members who saw a passion for improving engineering academia, timing systems, and the sport of competitive swimming as a whole. We extend our gratitude to everyone who was involved in the project!
Abstract

This project represents the culmination of an effort to redesign the timing system for competitive swim meets to record a swimmer’s precise time to the thousandth of a second, eclipsing the current standard competition precision tenfold. This required a redesign of the traditional swimming pool timing system to account for minute variations in the size of the pool and the length of the pool’s lanes. The scope of this project includes designing a system for obtaining precise measurement of the lengths along Swarthmore’s Ware Pool, the use of statistical analysis to determine the conformation of the pool’s lengths to NCAA (National Collegiate Athletics Association) standards, the selection of a detection method capable of recording one thousand measurements per second, and the implementation of this method in conjunction with a system capable of adjustments to account for the variation in pool/lane lengths and race distances. This project was implemented successfully, with results that indicate a precision three times greater than current timing systems, accounting for all possible sources of uncertainty or imprecision. Further suggested work includes the integration of this new system with pre-existing timing equipment at the pool and upgrading the camera system to obtain better resolution, data writing, and capture methods that were not possible with the given budget.
Introduction and Motivation

Currently, NCAA-sanctioned collegiate swim meets feature touchpad systems that use the pressure of a swimmer touching the wall to complete an electrical circuit which will stop the clock and record the swimmer’s time via a pressure plate. These results are reported to the hundredth of a second because slight variations in competition pool lengths make times reported to the thousandth of a second inaccurate. However, this means that swim races (which in the Centennial Conference can be as short as 19 seconds) more frequently result in ties than if the results could be made more accurate. As of this writing, the Centennial Conference record in the men’s 200 backstroke is jointly held by two swimmers of different teams, both with the same time recorded to the hundredth. One of the authors of this document (Freeman) has personally shared a joint first place in a race that ended in a tie at Ware Pool, Swarthmore’s home swimming facility.

Fig. 1 - A tie for third place at the 2024 Swarthmore Invitational in the 100-yard freestyle (47.26 seconds)
At the biggest stage for competitive swimming, the Olympics, ties have happened frequently. According to the International Olympic Committee, there have been 12 instances of ties at the Olympics, with one of the most notable being a **three way tie for second place** in the men’s 100 meter butterfly at the 2016 Summer Olympics.

![Fig. 2 - Michael Phelps, Chad Le Clos, and Laszlo Cseh share 2nd place in the 100-meter butterfly at the 2016 Rio Olympic Games.](image)

The most common reason for the frequency of ties is the fact that competitive swimming only records times to the hundredth of a second. From our research, it seems no electronic timing system accurate to the thousandth has ever been implemented in a competition pool. Only before automated timing systems did stopwatches record to the thousandth of a second. There, the likely lack of accurate measurement of these pool facilities and the inaccurate method of stopping the clock based on human reaction time meant that these results did not offer sufficient precision to be considered meaningful results.

It is our belief a redesign of the timing system that is able to take the exact length of the pool and its lanes into account that motivated us to address this problem at Swarthmore. Both participants working on the project have over 25 years of competitive swimming experience, and are very familiar with the state of current touchpad timing systems. With that much experience, both participants have been able to attend meets where it is very common to see a tie, especially in the shorter sprint events. Additionally, we believe our engineering backgrounds make us
uniquely qualified to tackle the specific requirements of this setup. Our dual foundations in structural analysis and design enabled us to prototype and safely implement a rail system to mount an overhead camera to record the swimmers as they finish a race. In addition, our specialized coursework in embedded systems (Freeman) and biomechanics (Wesley) allowed us to utilize this setup and employ appropriate object tracking and image analysis techniques to gather meaningful data on exactly when an athlete finishes a race in accordance with NCAA swimming standards for multiple strokes, in addition to helping us account for the unique motion of swimmers through water. Ben Freeman has also spent several years as an amateur astrophotographer, and is familiar with many of the camera communication protocols and image processing techniques that were employed over the course of this project. Ben Wesley also has spent a few months examining the materials of touchpads in preparation of a plan to design touchpads that are more accessible for paralympic swimmers. In short, both participants are steeped in the milieu of competitive swimming and possess the technical knowledge necessary to successfully bring the project to fruition.
Methods

The first goal of the project was to design and implement a camera setup that can be positioned so that the field of view of the camera encompasses the start/finish wall of Ware pool, including as many lanes as possible. Using a central wooden beam that extends up to the roof, the system was designed with as light a loading weight as possible to ensure the structural stability of the beam was not endangered.

**Fig. 3** - Initial Design for Location of Camera System in Swarthmore’s Ware Pool. The start/finish wall of the pool is located on the left side of this image, closest to the balcony. This was the diagram used to communicate key design elements to Project Manager Tom Cochrane which influenced how the system was secured to the overhead beam.

After several design iterations for capture methods, the camera selected for this project was the Sony RX100 IV, chosen for its unique intersection between 1000 FPS (frame per second) capture capability and affordability within our given budget. Due to the humid conditions of the pool, the camera could not remain fixed in place above the pool. Therefore, a system of two parallel unistrut rails has been used in conjunction with a system of vertical prime
angle (unistrut) supports connected to Ware Pool’s central roof beam to roll out a wooden cart that will hold the camera setup, all of which was synthesized from unistrut materials available in the machine shop, alleviating the stress on our budget. The load placed on the roof via the beam of the self-weight of the camera, support, cart, and track was found to be within acceptable limits as verified by the as-built schematics of Ware Pool’s roof, obtained from the College records office courtesy of Tom Cochran. From the measurements for the slope of the beam, required length of the horizontal unistrut rails and required length for the vertical supports were determined with the goals of the camera’s successful capture of a finish within a single lane in mind.

The unistrut track and supports were cut to the desired length, and fastened together using unistrut brackets and appropriately sized bolts. With the help of facilities, the vertical supports were suspended by attaching an L-bracket to a previously installed bolt along the beam. This required the removal of the 2 additional wire supports from the track, and a readjustments of where each of the two outside supports fall to align with the horizontal separation of the bolts in the beam. However, it is not believed that this had a measurable effect on the structural integrity of the camera capture setup due to the relatively light weight applied by the camera and the rolling cart. Additionally, facilities requested another design iteration that called for a larger number of holes in both the parallel 1x2 (now 1x4) connector plates for the track and the vertical L-bracket connections. Some of these connections required the assistance of J Johnson to weld custom connector plates as the desired hardware was not available in the shop.
Figs. 4, 5, & 6 - Cutting and Screwing the Unistrut Members together; Installation of the Vertical Supports for the Camera System
An additional bolt at each end of the track ensured the cart did not roll off the end of the track and into the pool. Using a string pulley system as well as a solid dowel, the camera cart was guided out to the end of the track such that the camera would overlook past the gutter of the swimming pool.

The tracks were installed on either side of the central beam overlooking the center of the pool. The camera cart was primarily responsible for maintaining the appropriate distance between the left side and right sides of the beam, although unistrut spacers were added that enforced a uniform length at the front and end of the track, on the outsides of the vertical supports. The vertical supports were connected to the central beam on either side and connected to the unistrut through the U-brackets bolted to the track. It was decided that the additional cable supports that were planned to be installed in the center of the beam were not used. In the end, there were four vertical supports across the beam; two at one end of the track, and two at the other end.

![Fig. 7 - The Finished Camera Frame](image-url)
Once this setup was installed, the camera, with the help of a controller aided to start and stop recording, collected our swimming data via triggering the controller on top of a ladder. This became necessary as it was discovered that the camera was only capable of shooting 4 seconds of 1000 fps video consecutively. The camera was then wheeled back over the balcony as the recorded video was written to the SD card in the camera and then transferred to a laptop to verify there were no focus or capture errors. There were five total trials that were conducted. The first four were all one-handed freestyle finishes. The first trial was a freestyle finish at a smooth speed to view the clearest possible finish at the slowest speed. From there, we did a trial at a moderate pace. Next, two trials were conducted at a simulated “50 freestyle sprint” speed. This pace was done twice as it is the closest speed to a realistic race in a competitive setting. Finally, an additional trial was carried out. In competitive swimming, two of the four official strokes require a two-handed touch on every wall including the finish: butterfly and breaststroke. To test the capabilities of our camera and the software we will be using, a trial of butterfly was done.

After all trials were complete, the five video clips were uploaded onto a computer for analysis. MATLAB was the software used for the motion tracking and image detection portion of the project, using the Mathworks multi-object tracking example as the foundation for our Kalman filter to detect motion between frames and later track the hand as it approaches the wall. From here, it was confirmed that a “flag region” could be set at a desired (and variable) length from the wall depending on the requirements of the race being competed. Once the parameters were adjusted to be optimized for the setup in which all the captures were taken (Ware Pool lane 3, at a measured distance of 22.861±0.003 m), the motion tracking of the hand was observed and the algorithm was tightened to better align with the specific needs of the hand touching the wall in our pixel region. This was completed for each of the five trials. From there, it was confirmed
that a sample timer (because the implementation of a real timer would require communication with the starter system already in place for college meets and fell outside the scope of this project) was able to be synchronized to the 1000 FPS footage, and could be implemented in future iterations of the project if the work were to continue.
Design Standards

One of the main governing standards within our design has to do with the official rules on what constitutes a “finish” in competitive swimming. Every year, USA Swimming releases a detailed rulebook on all of the technical mandates that encompass the start, turn, and finish of each stroke. The following constitutes as a legal freestyle finish:

“Finish - The swimmer shall have finished the race when any part of the individual’s body touches the wall after completing the predescribed race” (101.5.4 “FREESTYLE” USA Swimming, 2024)

It comes as no surprise that coordinating the fingertip to touch the wall is the fastest way to finish a race. The same rule applies in the stroke of butterfly, however both hands need touch the wall for a legal finish:

“Finish - At the finish, the body shall be on the breast and touch shall be made with both hands separated and simultaneously at, above, or below the water surface.” (101.3.4 “BUTTERFLY” USA Swimming, 2024)

With these considerations in mind, we applied all of the rules and regulations laid out by USA Swimming into our data to ensure the project did not run afoul of any possible rules governing a competitive swimming finish, particularly those around simultaneity for breaststroke and butterfly strokes. It was ultimately decided that the image tracking program is, at the time of
this writing, not robust enough to flag illegal one-hand touches, and this disqualification, as is the case in touchpad timing systems, would still have to be flagged by the meet officials.

Additionally, in order for the camera frame to be installed and hang of a wooden beam along Ware Pool’s ceiling, allowable building stress had to be considered. Having a large camera frame suspended on a central beam of a ceiling at a pool with many Swarthmore College community members using the facility enforced safety considerations for those around it. After informing Project Manager of Facilities Tom Cochrane of the total weight the frame would bear on the ceiling (approximately two hundred pounds), it was determined that it would not affect the structural integrity of the building in any form.
Results

Prior to assembling the timing arm, a statistical analysis of the pool found that the deviation between the lengths of the lanes was not significant enough to warrant differentiation on the scale of 1 mm. For context, a competition 25-yard pool should be 22.860 meters, to the closest mm. The lane in which the camera was tested, lane 3, was measured to be the correct length to within 1 mm, and given its length was likely to be longer than the standard distance of the pool (and therefore ideal for setting a “flag region” before the end of the lane) it was chosen to be timed in the analysis portion of this project to the end of the wall. Although our designed system can measure to the thousandth and will account for lanes that are built 3 mm or longer from the standard length, this method will also increase the accuracy of the timing by eliminating the shortened length of the pool due to the insertion of a touchpad, which removes up to 1 inch of length from the lane depending on the model and number of touchpads a facility uses.

<table>
<thead>
<tr>
<th>Lane #</th>
<th>Trial 1 length (m)</th>
<th>Trial 2 length (m)</th>
<th>Trial 3 length (m)</th>
<th>Average length (m)</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>22.86</td>
<td>22.859</td>
<td>22.865</td>
<td>22.861</td>
<td>0.003</td>
</tr>
<tr>
<td>1</td>
<td>22.858</td>
<td>22.856</td>
<td>22.865</td>
<td>22.860</td>
<td>0.005</td>
</tr>
<tr>
<td>2</td>
<td>22.852</td>
<td>22.857</td>
<td>22.857</td>
<td>22.857</td>
<td>0.003</td>
</tr>
<tr>
<td>3</td>
<td>22.862</td>
<td>22.861</td>
<td>22.856</td>
<td>22.861</td>
<td>0.003</td>
</tr>
<tr>
<td>4</td>
<td>22.859</td>
<td>22.86</td>
<td>22.861</td>
<td>22.860</td>
<td>0.001</td>
</tr>
<tr>
<td>5</td>
<td>22.862</td>
<td>22.861</td>
<td>22.86</td>
<td>22.861</td>
<td>0.001</td>
</tr>
<tr>
<td>6</td>
<td>22.86</td>
<td>22.86</td>
<td>22.859</td>
<td>22.860</td>
<td>0.001</td>
</tr>
</tbody>
</table>
Additionally, it was found that the pace of the pool record 50 freestyle (the fastest competition event) did not eclipse the 3mm mark as the minimum resolution attainable by our measurements of the pool. The point up to which we could reliably measure the length of the pool, 3mm, was compliant with the speed at which the swimmers at Swarthmore are able to compete, as seen in Table 4. The accuracy of our measurement practices did not prohibit any race from being imaged, as the fastest 50 free ever swam in Ware Pool did not attain an average speed greater than 3 mm/0.001 seconds, confirming the uncertainty in the pool length as the controlling uncertainty of our project. Additional accuracy can be obtained in the future by employing more precise measurement equipment when obtaining the length of the pool lanes, or by adapting the system to be calibrated to multiple lanes so their individual deviations from the standard can be taken into account. This method, however, will likely require a camera with a wider field of view and more capture flexibility than the RX 100 IV is capable of.

Table 1. - Distances of each lane of Ware Pool using the Laser Measurer Device.

<table>
<thead>
<tr>
<th></th>
<th>22.863</th>
<th>22.862</th>
<th>22.866</th>
<th>22.863</th>
<th>0.002</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>22.864</td>
<td>22.863</td>
<td>22.861</td>
<td>22.86266667</td>
<td>0.002</td>
</tr>
<tr>
<td>9</td>
<td>22.864</td>
<td>22.866</td>
<td>22.866</td>
<td>22.86533333</td>
<td>0.001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mean Distance Value</th>
<th>22.861</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median Value</td>
<td>22.86</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>22.852</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>22.866</td>
</tr>
<tr>
<td>First Quartile</td>
<td>22.859</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
</tr>
<tr>
<td>Third Quartile</td>
<td>22.863</td>
</tr>
<tr>
<td>Estimated Uncertainty</td>
<td>0.00061</td>
</tr>
</tbody>
</table>

**Table 2.** - Mean Distances of all Lanes

<table>
<thead>
<tr>
<th>Null Hypothesis:</th>
<th>Both halves of pool are the same length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean distance s1 (0-4)</td>
<td>0.002268136778</td>
</tr>
<tr>
<td>Mean distance s2 (5-9)</td>
<td>0.002218608172</td>
</tr>
<tr>
<td>1st set degree of freedom</td>
<td>14</td>
</tr>
<tr>
<td>2nd set degree of freedom</td>
<td>14</td>
</tr>
<tr>
<td>Test Statistic, F</td>
<td>1.022</td>
</tr>
</tbody>
</table>

**Table 3.** - Null Hypothesis for Length of each Pool Lane
Table 4. - Verification of timing system feasibility. Given the former 50-yard freestyle pool record time (20.60 seconds) and an assumption that the lane was exactly 25 yards (75 ft), these are the expected distances traveled in 0.01 seconds and 0.001 seconds as well as how much time it would take to travel unit distances of 1 cm and 1 mm, proving that our method is capable of capturing a 50-yard freestyle within Ware Pool to 0.0001 seconds when accounting for the varying lengths of the lane, by a factor of about 3. Please note that this table does not take into account the uncertainty in the laser measurement device or the resolving depth of the camera.

<table>
<thead>
<tr>
<th>Time Interval</th>
<th>Distance traveled in record 50 freestyle (average speed)</th>
<th>Converted distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01 sec</td>
<td>0.901 cm</td>
<td>0.02956ft</td>
</tr>
<tr>
<td>0.001 sec</td>
<td>0.90 mm</td>
<td>0.002956ft</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Distance Interval</th>
<th>Time taken in record 50 freestyle (average speed)</th>
<th>--</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 cm</td>
<td>0.0011 sec</td>
<td>--</td>
</tr>
<tr>
<td>1 mm</td>
<td>0.00011 sec</td>
<td>--</td>
</tr>
</tbody>
</table>

Before any of the race finish data were collected, our group wanted to do a trial-run of the camera’s motion tracking capabilities. A test recording was done by having a subject wave their hand outward into the camera’s frame to simulate a race finish as viewed from above. Once the camera recorded the clip in slow motion and in the one-thousand frames per second mode, the algorithm was able to track the fingertips of the subject as it moved across the width of the doorframe. Below is a screenshot of the predicted motion of the subject’s hand. Minimal
erroneous predicted motion boxes were detected, and the primary motion tracker (the one highlighted below) never experienced interruptions in its motion path.

![Image of motion tracker](image.png)

**Fig. 8** - Test “Hand” Trial analyzed under Matlab’s “Motion Based Multi-Object Tracking” Code

Most of the results were analyzed using the aforementioned MATLAB script (Appendix A) pertaining to motion-based object tracking. Motion tracking results were applied to each of the five swimming clips. Additionally, a “pseudo-timer” was added to each video to simulate the exact moment the swimmer finished into the wall.
Fig. 9 - Trial “Free_1” (the first trial of slower, smooth, freestyle stroke) analyzed thorough Matlab’s “Motion Based Multi-Object Tracking” Code

Most of the videos required no substantial alterations with the size of the predicted motion box. This made it easy to track the leading box which would ultimately include the fingertip that broke the plane of the flag region first, signifying the end of the race for the purposes of this project. A limitation of this system was found in the required employment of custom cropping to reduce excess noise picked up by the motion tracking script as a result of the splashing produced by a full-speed finish. This cropping was different for each trial analyzed, but since the trials needed to be individually analyzed anyway to determine the flag region, this did not pose too much of an obstacle to the efficiency of the process as it stands currently.
Discussion

On the whole, the successful implementation of this project demonstrates the feasibility of possible timing systems in the future capable of capturing race metrics with additional accuracy when compared to current standards. This suggests new avenues for novel timing systems at high levels of competition, including the upcoming Summer Olympic Games held in Paris, France. Additional trials in freestyle as well as the other three competitive strokes (butterfly, backstroke, and breaststroke) would be needed for further analysis to examine accuracy in depth of the hand finish as well as the capabilities of recording two-handed finishes seen in butterfly and breaststroke. Currently, this group has no method of obtaining a resolving distance, the smallest distinction the camera was able to observe at a distance of the height of the system above the water. This represents a major area of uncertainty that can be ameliorated through either greater resolution or by placing a measuring tool or object of known length in the frame of another trial. By the time this analysis was conducted, the suspended camera system had been deconstructed in order to return its materials to their owners. Although “noisy” or splashy data was at a minimum in our experiment, it is to be expected that at larger competitive
swim meets, there will be much more apparent water turbulence. This is expected to remain a hurdle if the project were to be continued, as individual analysis currently requires the usage of a custom crop for every finish for the purposes of noise reduction. In this case, a higher quality camera can go a long way toward visual improvement, noise reduction, and by extension workflow efficiency augmentation through the removal of cropping as a step in the process.

In our butterfly trial, the motion-tracking code was able to stop the exact moment (within our uncertainties) the hand was within three millimeters of the wall. This was a trial that our group was initially worried about because part of the starting block impedes the visuals of the swimmer’s left hand as he finishes into the wall. However our group imposed the assumption that as soon as the right hand touched the wall, the left would do so as well; abiding by the constraints established by the USA Swimming stroke and turn rulebook as outlined in the “Standards” section.

Reflecting back on our initial statistical analysis of the length of Ware Pool, it was important to make sure all lanes of the pool conform to the same distance such that our experiment could have hypothetically been done in any lane. Table 4 illustrates the distance traveled in a hundredth and a thousandth of a second under the constraint of the former 50-yard freestyle Ware Pool record. After establishing the relative speeds of the hand at the finish, it was determined that an uncertainty of 3 mm is allowable. This means that one thousandth of a second is larger than the time it would take for the hand to travel 3 mm at the finish. This number can be skewed however, as the speed of the former pool record used was averaged out among the entire 50-yard distance. When in actuality, the speed of the swimmer changes multiple times;
specifically on the start and flipturn. Future work for the project in this area may include the implementation of more precise equipment for measuring the length of the lanes, thus reducing the uncertainty in the lanes and positively impacting the “resolution limit” set currently at 3 mm.

Figures 8-9 highlight the results of the process of tuning the motion tracking algorithm to most closely fit the furthest hand forward in freestyle and butterfly finishes as the swimmer crosses the flag region. As can be seen in the figures, there is generally a tight agreement between the software’s understanding of where the hand is and the hand’s actual position in the footage. This was observed for all trials with slight parameter adjustment in the flag regions and cropping for noise reduction. Even with the poor resolution obtained through the videos and the limited functionality of the camera (4s maximum HFR/high frame rate capture, no wireless capture integration, video resolution capped at 800x270, etc.) these are remarkably encouraging results, and seems to support the hypothesis that ties can be reduced in swimming by accounting for the variation in pool and lane lengths with slight design improvements and wider adoption. In Centennial Conference competition this year, a capture system similar to our design may have resolved as many as 8 individual and relay ties over the course of the season, including one that required an additional swim-off race in the Conference championship meet.

The challenges faced over the course of the execution of this project were myriad. To begin, the design required several iterations to settle on the final product, with changes in length, number of tracks, number of supports, and method of analysis present throughout the duration based on feedback on structural feasibility, equipment limitations, and input from Facilities on installation logistics. In addition, the tight budget necessitated the employment of a low-end camera (optimizing the ability to record 1000 FPS as opposed to having to write an algorithm
that could reliably interpolate frames) and borrowed unistrut from the machine shop. This placed practical constraints on the project, from the limitations on the camera discussed above to the slight inaccuracies that were introduced as a result of having to marry two pieces of unistrut together when one single piece (which was not available as part of the shop’s stock) would have resulted in a smoother and more accurate fit. As a result of these limitations, the capture method was significantly less abbreviated than we were hoping, but it is believed with a larger budget of $1500 and more time to complete testing, a more effective resolution, accuracy, and capture method can be attained.

**Conclusion**

It can be concluded that the use of a motion-based tracking aide can offer a more precise competitive timer not only at Ware Pool, but in all pools that hold competitive swim meets. As mentioned above, our results allow us to conclude that the use of this new method is roughly three times more precise than the use of touchpads. The hope is that in the near future, Fédération internationale de natation (FINA), the head governing body of world aquatics, will accept this new technology and implement it similar to that of the use of slit cameras in Olympic track races. Use of Video demos of the motion-tracking code were included in the final presentation for visual demonstrations of the final product. A few takeaways from what we learned was that the resolution of the camera can assist the motion-tracking code, improving the precision of recorded times greatly.
Bibliography


- USA Swimming, “2024 USA Swimming Rules and Regulations” *USA SWIMMING, INC.* 2024, pp. 23-24

Appendix

APPENDIX (A): MATLAB Code

- (A.1):
  clc;    % Clear the command window.
close all; % Close all figures (except those of imtool.)
imtool close all; % Close all imtool figures.
clear; % Erase all existing variables.
workspace; % Make sure the workspace panel is showing.
fontSize = 14;
obj = VideoReader('E90TestVideo.mp4');
vid = read(obj);
frames = obj.NumFrames;
for x = 1 : frames
    imwrite(vid(:,:,,:),strcat('frame-',num2str(x),'.png'));
end

- (A.2):
clear
close all
obj = VideoReader('E90 Test Video.mp4');
vid = read(obj);
frames = obj.NumFrames;
writer = VideoWriter("New E90 Test Video p1.mp4","MPEG-4");
open(writer);
for x = 1:frames
    imwrite(vid(:,:,,:),strcat('frame-',num2str(x),'.png'));
    I = imread(strcat('frame-',num2str(x),'.png'));
    position = [100 1000];
    sec = x*0.001;
    value = strcat(num2str(sec));
    RGB = insertText(I,position,value,AnchorPoint="LeftBottom",FontSize=30);
writeVideo(writer,RGB)
end
clear reader
close(writer)

- (A.3):

%% E90 Project Part 2
% This code utilizes the Motion Based Multi-Object Tracking Example
% provided by MATLAB to track Kate Hallmark's hand in the doorway. It also
% keeps track of the brightness in a denoted area that the hand passes
% through. Once the hand obscures a considerable amount of light (i.e., the
% brightness decreases by more than 2.5% of the original brightness), then
% a variable "compare" shifts from 0 to 1. This feedback is given in the
% command window.
% Create System objects used for reading video, detecting moving objects,
% and displaying the results.
function E14E90projpart2()
clc
close all
import vision.*
% Detect moving objects, and track them across video frames.
obj = setupSystemObjects();
tracks = initializeTracks(); % Create an empty array of tracks.
nextId = 1; % ID of the next track
nframe = 0;
flagv = 0;
% Detect moving objects, and track them across video frames.
while hasFrame(obj.reader)
    nframe = nframe + 1; % Define a frame counter
    frame = readFrame(obj.reader);
imrect = [200 0 700 1080];
frame = imcrop(frame,imrect);
[centroids, bboxes, mask] = detectObjects(frame);
predictNewLocationsOfTracks();
[assignments, unassignedTracks, unassignedDetections] = ...
    detectionToTrackAssignment();
    % Defining a flag variable
    rightpos = bboxes(:,1) + bboxes(:,3); % Find the rightmost bound of the
        % bounding box; to find the lowermost bound, use bboxes(:,2) + bboxes(:,4)
    rightbound = 300; % Define the threshold as crossing the line x =
    % 300 (in pixels) - ExtractMovieFrames plots images with tickmarks on x and y
    axes to more easily count pixels
    if rightpos > rightbound
        flagv = 1; % Define a flag variable that switches from 0 to 1
    end
    updateAssignedTracks();
updateUnassignedTracks();
deleteLostTracks();
createNewTracks();
displayTrackingResults();

fprintf('Processed frame %4d. Flag var = %d\n', nframe, flagv); % Update
the user on frame number and flag variable value
end

%% Create System Objects
% Create System objects used for reading the video frames, detecting
% foreground objects, and displaying results.
function obj = setupSystemObjects()
% Initialize Video I/O
% Create objects for reading a video from a file, drawing the tracked
% objects in each frame, and playing the video.
% Create a video reader.
obj.reader = VideoReader('New E90 Test Video p1.mp4');
% Create two video players, one to display the video,
% and one to display the foreground mask.
obj.maskPlayer = vision.VideoPlayer('Position', [0, 0, 700, 1080]);
obj.videoPlayer = vision.VideoPlayer('Position', [900, 0, 700, 1080]);
% Create System objects for foreground detection and blob analysis
% The foreground detector is used to segment moving objects from
% the background. It outputs a binary mask, where the pixel value
% of 1 corresponds to the foreground and the value of 0 corresponds
% to the background.
obj.detector = vision.ForegroundDetector('NumGaussians', 3, ...
    'NumTrainingFrames', 10, 'MinimumBackgroundRatio', 0.8);%
% Connected groups of foreground pixels are likely to correspond to moving
% objects. The blob analysis System object is used to find such groups
% (called 'blobs' or 'connected components'), and compute their
% characteristics, such as area, centroid, and the bounding box.
obj.blobAnalyser = vision.BlobAnalysis('BoundingBoxOutputPort', true, ...
    'AreaOutputPort', true, 'CentroidOutputPort', true, ...
    'MinimumBlobArea', 400);
end

%% Initialize Tracks
% The |initializeTracks| function creates an array of tracks, where each
% track is a structure representing a moving object in the video. The
% purpose of the structure is to maintain the state of a tracked object.
% The state consists of information used for detection to track assignment,
% track termination, and display.
% The structure contains the following fields:
% *
% |id| : the integer ID of the track
% |bbox| : the current bounding box of the object; used
%    for display
% |kalmanFilter| : a Kalman filter object used for motion-based
tracking
% |age| : the number of frames since the track was first detected
% |totalVisibleCount| : the total number of frames in which the track was detected (visible)
% |consecutiveInvisibleCount| : the number of consecutive frames for which the track was not detected (invisible).

Noisy detections tend to result in short-lived tracks. For this reason, the example only displays an object after it was tracked for some number of frames. This happens when |totalVisibleCount| exceeds a specified threshold.

When no detections are associated with a track for several consecutive frames, the example assumes that the object has left the field of view and deletes the track. This happens when |consecutiveInvisibleCount| exceeds a specified threshold. A track may also get deleted as noise if it was tracked for a short time, and marked invisible for most of the frames.

```matlab
function tracks = initializeTracks()
% create an empty array of tracks
tracks = struct(...
    'id', {}, ...
    'bbox', {}, ...
    'kalmanFilter', {}, ...
    'age', {}, ...
    'totalVisibleCount', {}, ...
    'consecutiveInvisibleCount', {});
end
```

Detect Objects
The |detectObjects| function returns the centroids and the bounding boxes of the detected objects. It also returns the binary mask, which has the same size as the input frame. Pixels with a value of 1 correspond to the foreground, and pixels with a value of 0 correspond to the background.

The function performs motion segmentation using the foreground detector. It then performs morphological operations on the resulting binary mask to remove noisy pixels and to fill the holes in the remaining blobs.

```matlab
function [centroids, bboxes, mask] = detectObjects(frame)
% Detect foreground.
mask = obj.detector.step(frame);
% Apply morphological operations to remove noise and fill in holes.
mask = imopen(mask, strel('rectangle', [3,3]));
mask = imclose(mask, strel('rectangle', [15, 15]));
mask = imfill(mask, 'holes');
% Perform blob analysis to find connected components.
[~, centroids, bboxes] = obj.blobAnalyser.step(mask);
end
```
%% Predict New Locations of Existing Tracks
% Use the Kalman filter to predict the centroid of each track in the
% current frame, and update its bounding box accordingly.
function predictNewLocationsOfTracks()
    for i = 1:length(tracks)
        bbox = tracks(i).bbox;
        % Predict the current location of the track.
        predictedCentroid = predict(tracks(i).kalmanFilter);
        % Shift the bounding box so that its center is at
        % the predicted location.
        predictedCentroid = int32(predictedCentroid) - bbox(3:4) / 2;
        tracks(i).bbox = [predictedCentroid, bbox(3:4)];
    end
end

%% Assign Detections to Tracks
% Assigning object detections in the current frame to existing tracks is
% done by minimizing cost. The cost is defined as the negative
% log-likelihood of a detection corresponding to a track.
% The algorithm involves two steps:
% Step 1: Compute the cost of assigning every detection to each track using
% the |distance| method of the |vision.KalmanFilter| System object(TM). The
% cost takes into account the Euclidean distance between the predicted
% centroid of the track and the centroid of the detection. It also includes
% the confidence of the prediction, which is maintained by the Kalman
% filter. The results are stored in an MxN matrix, where M is the number of
% tracks, and N is the number of detections.
% Step 2: Solve the assignment problem represented by the cost matrix using
% the |assignDetectionsToTracks| function. The function takes the cost
% matrix and the cost of not assigning any detections to a track.
% The value for the cost of not assigning a detection to a track depends on
% the range of values returned by the |distance| method of the
% |vision.KalmanFilter|. This value must be tuned experimentally. Setting
% it too low increases the likelihood of creating a new track, and may
% result in track fragmentation. Setting it too high may result in a single
% track corresponding to a series of separate moving objects.
% The |assignDetectionsToTracks| function uses the Munkres' version of the
% Hungarian algorithm to compute an assignment which minimizes the total
% cost. It returns an M x 2 matrix containing the corresponding indices of
% assigned tracks and detections in its two columns. It also returns the
% indices of tracks and detections that remained unassigned.
function [assignments, unassignedTracks, unassignedDetections] = ...
    detectionToTrackAssignment()
    nTracks = length(tracks);
nDetections = size(centroids, 1);
% Compute the cost of assigning each detection to each track.
cost = zeros(nTracks, nDetections);
for i = 1:nTracks
    cost(i, :) = distance(tracks(i).kalmanFilter, centroids);
end
% Solve the assignment problem.
costOfNonAssignment = 30;
[assignments, unassignedTracks, unassignedDetections] = ...
    assignDetectionsToTracks(cost, costOfNonAssignment);
end

%% Update Assigned Tracks
% The |updateAssignedTracks| function updates each assigned track with the
% corresponding detection. It calls the |correct| method of
% |vision.KalmanFilter| to correct the location estimate. Next, it stores
% the new bounding box, and increases the age of the track and the total
% visible count by 1. Finally, the function sets the invisible count to 0.
function updateAssignedTracks()
    numAssignedTracks = size(assignments, 1);
    for i = 1:numAssignedTracks
        trackIdx = assignments(i, 1);
        detectionIdx = assignments(i, 2);
        centroid = centroids(detectionIdx, :);
        bbox = bboxes(detectionIdx, :);
        % Correct the estimate of the object's location
        % using the new detection.
        correct(tracks(trackIdx).kalmanFilter, centroid);
        % Replace predicted bounding box with detected
        % bounding box.
        tracks(trackIdx).bbox = bbox;
        % Update track's age.
        tracks(trackIdx).age = tracks(trackIdx).age + 1;
        % Update visibility.
        tracks(trackIdx).totalVisibleCount = ...
            tracks(trackIdx).totalVisibleCount + 1;
        tracks(trackIdx).consecutiveInvisibleCount = 0;
    end
end

%% Update Unassigned Tracks
% Mark each unassigned track as invisible, and increase its age by 1.
function updateUnassignedTracks()
    for i = 1:length(unassignedTracks)
        ind = unassignedTracks(i);
        tracks(ind).age = tracks(ind).age + 1;
        tracks(ind).consecutiveInvisibleCount = ...
            tracks(ind).consecutiveInvisibleCount + 1;
    end
end

%% Delete Lost Tracks
The `deleteLostTracks` function deletes tracks that have been invisible for too many consecutive frames. It also deletes recently created tracks that have been invisible for too many frames overall.

```matlab
function deleteLostTracks()
    if isempty(tracks)
        return;
    end
    invisibleForTooLong = 20;
    ageThreshold = 12;
    % Compute the fraction of the track's age for which it was visible.
    ages = [tracks(age)];
    totalVisibleCounts = [tracks(totalVisibleCount)];
    visibility = totalVisibleCounts ./ ages;
    % Find the indices of 'lost' tracks.
    lostInds = (ages < ageThreshold & visibility < 0.6) | ... % tracks.consecutiveInvisibleCount] >= invisibleForTooLong;
    % Delete lost tracks.
    tracks = tracks(~lostInds);
end
```

%% Create New Tracks
Create new tracks from unassigned detections. Assume that any unassigned detection is a start of a new track. In practice, you can use other cues to eliminate noisy detections, such as size, location, or appearance.

```matlab
function createNewTracks()
    centroids = centroids(unassignedDetections, :);
    bboxes = bboxes(unassignedDetections, :);
    for i = 1:size(centroids, 1)
        centroid = centroids(i,:);
        bbox = bboxes(i, :);
        % Create a Kalman filter object.
        kalmanFilter = configureKalmanFilter('ConstantVelocity', ... % centroid, [200, 50], [100, 25], 100);
        % Create a new track.
        newTrack = struct(...
            'id', nextId, ...
            'bbox', bbox, ...
            'kalmanFilter', kalmanFilter, ...
            'age', 1, ...
            'totalVisibleCount', 1, ...
            'consecutiveInvisibleCount', 0);
        % Add it to the array of tracks.
        tracks(end + 1) = newTrack;
        % Increment the next id.
        nextId = nextId + 1;
    end
end
```

%% Display Tracking Results
The `displayTrackingResults` function draws a bounding box and label ID for each track on the video frame and the foreground mask. It then
function displayTrackingResults()
% Convert the frame and the mask to uint8 RGB.
frame = im2uint8(frame);
mask = uint8(repmat(mask, [1, 1, 3]) .* 255);
minVisibleCount = 8;
if ~isempty(tracks)
    % Noisy detections tend to result in short-lived tracks.
    % Only display tracks that have been visible for more than
    % a minimum number of frames.
    reliableTrackInds = ... [tracks(:,totalVisibleCount] > minVisibleCount;
    reliableTracks = tracks(reliableTrackInds);
    % Display the objects. If an object has not been detected
    % in this frame, display its predicted bounding box.
    if ~isempty(reliableTracks)
        % Get bounding boxes.
        bboxes = cat(1, reliableTracks.bbox);
        % Get ids.
        ids = int32([reliableTracks(:,id])];
        % Create labels for objects indicating the ones for
        % which we display the predicted rather than the actual
        % location.
        labels = cellstr(int2str(ids'));
        predictedTrackInds = ... [reliableTracks(:,consecutiveInvisibleCount] > 0;
        isPredicted = cell(size(labels));
        isPredicted(predictedTrackInds) = {' predicted'};
        labels = strcat(labels, isPredicted);
        % Draw the objects on the frame.
        frame = insertObjectAnnotation(frame, 'rectangle', ... bboxes, labels);
        % Draw the objects on the mask.
        mask = insertObjectAnnotation(mask, 'rectangle', ... bboxes, labels);
    end
end
%
% Display the mask and the frame.
%obj.maskPlayer.step(mask);
%obj.videoPlayer.step(frame);
end
end

% ExtractMovieFrames:

% Demo to extract frames and get frame means from a movie and save individual
% frames to separate image files.
% Then rebuilds a new movie by recalling the saved images from disk.
% Also computes the mean gray value of the color channels
% And detects the difference between a frame and the previous frame.
% Illustrates the use of the VideoReader and VideoWriter classes.
% A Mathworks demo (different than mine) is located here
%
clc;  % Clear the command window.
close all;  % Close all figures (except those of imtool.)
clear;  % Erase all existing variables.
workspace;  % Make sure the workspace panel is showing.
fontSize = 22;
compare = 0;
% Initialize options.
wantsFrameStamps = false;
wantsSameSize = true;
writeToDisk = false;
outputMovieFullFileName = [];
% Open the rhino.avi demo movie that ships with MATLAB.
% First get the folder that it lives in.
folder = fileparts(which('E90 Test Video.mp4')); % Determine where demo
folder is (works with all versions).
% Pick one of the two demo movies shipped with the Image Processing Toolbox.
% Comment out the other one.
inputMovieFullFileName = fullfile(folder, 'E90 Test Video.mp4');
% movieFullFileName = fullfile(folder, 'traffic.avi'); % An alternate demo movie.
% Check to see that this movie file actually exists.
if ~exist(inputMovieFullFileName, 'file')
    strErrorMessage = sprintf('File not found:
%You can choose a new one, or cancel', inputMovieFullFileName);
    response = questdlg(strErrorMessage, 'File not found', 'OK - choose a new movie.', 'Cancel', 'OK - choose a new movie.);
    if strcmpi(response, 'OK - choose a new movie.')
        [baseFileNameNoExt, folderName, FilterIndex] = uigetfile('*\*.avi');
        if ~isequal(baseFileNameNoExt, 0)
            inputMovieFullFileName = fullfile(folderName, baseFileNameNoExt);
        else
            return;
        end
    else
        return;
    end
end
try
% Open up a VideoReader object to read in the frames from the existing movie.
videoReaderObject = VideoReader(inputMovieFullFileName);
% Determine how many frames there are.
numberOfFrames = videoReaderObject.NumFrames;
vidHeight = videoReaderObject.Height;
vidWidth = videoReaderObject.Width;

numberOfFramesWritten = 0;
% Prepare a figure to show the images in the upper half of the screen.
fig = figure('Name', 'Video Demo by Image Analyst', 'NumberTitle', 'Off');
% screenSize = get(0, 'ScreenSize');
% Enlarge figure to full screen.
% set(gcf, 'Units', 'Normalized', 'OuterPosition', [0, 0.04, 1, 0.96]); % Old style
fig.WindowState = 'maximized'; % New way of maximizing.

% Loop through the movie, writing all frames out.
% Each frame will be in a separate file with unique name.
meanGrayLevels = zeros(numberOfFrames, 1);
meanRedLevels = zeros(numberOfFrames, 1);
meanGreenLevels = zeros(numberOfFrames, 1);
meanBlueLevels = zeros(numberOfFrames, 1);
for frame = 1 : numberOfFrames
    % Extract the frame from the movie structure.
    thisFrame = read(videoReaderObject, frame);
    imrect = [300 300 200 200];
    thisFrame = imcrop(thisFrame,imrect);
    % Display it
    hImage = subplot(2, 2, 1);
    image(thisFrame);
    caption = sprintf('Frame %4d of %d.', frame, numberOfFrames);
    title(caption, 'FontSize', fontSize);
    axis('on', 'image'); % Show tick marks and get aspect ratio correct.
    drawnow; % Force it to refresh the window.

    % Calculate the mean gray level.
    grayImage = rgb2gray(thisFrame);
meanGrayLevels(frame) = mean(grayImage(:));

% Calculate the mean R, G, and B levels.
meanRedLevels(frame) = mean(mean(thisFrame(:, :, 1)));
meanGreenLevels(frame) = mean(mean(thisFrame(:, :, 2)));
meanBlueLevels(frame) = mean(mean(thisFrame(:, :, 3)));

% Plot the mean gray levels.
hPlot = subplot(2, 2, 2);
hold off;
plot(meanGrayLevels, 'k-', 'LineWidth', 3);
hold on;
plot(meanRedLevels, 'r-', 'LineWidth', 2);
plot(meanGreenLevels, 'g-', 'LineWidth', 2);
plot(meanBlueLevels, 'b-', 'LineWidth', 2);
ggrid on;

% Put title back because plot() erases the existing title.
title('Mean Intensities In Gray Levels', 'FontSize', fontSize);

if frame == 1
    xlabel('Frame Number');
    ylabel('Gray Level');
    [rows, columns, numberOfColorChannels] = size(thisFrame);
end

% Update user with the progress. Display in the command window.
if writeToDisk
    progressIndication = sprintf('Wrote frame %4d of %d.
                  Compare = %d', frame, numberOfFrames, compare);
else
    progressIndication = sprintf('Processed frame %4d of %d.
                  Compare = %d', frame, numberOfFrames, compare);
end
disp(progressIndication);

% Increment frame count (should eventually = numberOfFrames
% unless an error happens).
numberOfFramesWritten = numberOfFramesWritten + 1;

% Now let's do the differencing
alpha = 0.5;
if frame == 1
    Background = thisFrame;
else
    % Change background slightly at each frame.
    % Each time the background is a weighted average of the all
    % prior background frames
% with decreasing weights the further back in time the
% Background(t+1)=(1-alpha)*I+alpha*Background(t)
Background = (1-alpha)* thisFrame + alpha * Background;
end
% Display the changing/adapting background.
subplot(2, 2, 3);
imshow(Background);
title('Adaptive Background', 'FontSize', fontSize);
axis('on', 'image'); % Show tick marks and get aspect ratio correct.
% Calculate a difference between this frame and the background.
differenceImage = thisFrame - uint8(Background);
% Threshold with Otsu method.
grayImage = rgb2gray(differenceImage); % Convert to gray level
thresholdLevel = graythresh(grayImage); % Get threshold.
binaryImage = imbinarize( grayImage, thresholdLevel); % Do the
binarization
% Plot the binary image.
subplot(2, 2, 4);
imshow(binaryImage);
title('Binarized Difference Image', 'FontSize', fontSize);
axis('on', 'image'); % Show tick marks and get aspect ratio correct.
% set a variable to compare the brightness of the previous frame
gray0 = meanGrayLevels(1); % Find the initial brightness of the region
if frame == 1
    compare = 1;
elseif meanGrayLevels(frame-1) - gray0 < -0.025*gray0 % Check if the
    current brightness is 2.5% lower than original brightness
    compare = 1;
else
    compare = 0;
end
end
xlabel(hPlot, 'Frame Number', 'FontSize', fontSize);
ylabel(hPlot, 'Gray Level', 'FontSize', fontSize);
legend(hPlot, 'Overall Brightness', 'Red Channel', 'Green Channel',
'Blue Channel', 'Location', 'Northwest');
compare = 0;
index = 0;
%----------------------------------------------------------------------
% Alert user that we're done.
if writeToDisk
    finishedMessage = sprintf('Done! It wrote %d frames to folder\n%s', numberOfFramesWritten, outputFolder);
else
finishedMessage = sprintf('Done! It processed %d frames of
"%s"', numberOfFramesWritten, inputMovieFullFileName);
end
disp(finishedMessage); % Write to command window.
uiwait(msgbox(finishedMessage)); % Also pop up a message box.

% Exit if they didn't write any individual frames out to disk.
if ~writeToDisk
    return;
end
% Close old figure.
close(hFig);
end
for iframe = 1:numberOfFramesWritten
    if iframe == 1
        compare = 0;
        index = 0;
    else
        if meanGrayLevels(iframe) - meanGrayLevels(iframe-1) > 1 &&
            compare == 0 && index == 0
            compare = 1;
            index = iframe;
        end
    end
end

- (A.5)
Code used for butterfly trials

%% E90 Project Part 2 - Updated for Butterfly
% This code utilizes the Motion Based Multi-Object Tracking Example
% provided by MATLAB to track Kate Hallmark's hand in the doorway. It also
% keeps track of the brightness in a denoted area that the hand passes
% through. Once the hand obscures a considerable amount of light (i.e., the
% brightness decreases by more than 2.5% of the original brightness), then
% a variable "compare" shifts from 0 to 1. This feedback is given in the
% command window.
% Create System objects used for reading video, detecting moving objects,
% and displaying the results.
function E14E90projpart2()
    clc
close all
import vision.*
% Detect moving objects, and track them across video frames.
obj = setupSystemObjects();
tracks = initializeTracks(); % Create an empty array of tracks.
nextId = 1; % ID of the next track
nframe = 0;
flagv = 0;

% Detect moving objects, and track them across video frames.
while hasFrame(obj.reader)
    nframe = nframe + 1; % Define a frame counter
    frame = readFrame(obj.reader);
    imrect = [700 0 800 500];
    frame = imcrop(frame, imrect);
    [centroids, bboxes, mask] = detectObjects(frame);
    predictNewLocationsOfTracks();
    [assignments, unassignedTracks, unassignedDetections] = ... detectionToTrackAssignment();
    % Defining a flag variable
    lowpos = max(bboxes(:, 2) + bboxes(:, 4)); % Find the rightmost bound of the bounding box; to find the lowermost bound, use bboxes(:, 2) + bboxes(:, 4)
    lowbound = 397; % Define the threshold as crossing the line x = 300 (in pixels) - ExtractMovieFrames plots images with tickmarks on x and y axes to more easily count pixels
    if lowpos > lowbound
        flagv = 1; % Define a flag variable that switches from 0 to 1 when the threshold is crossed
    end

    updateAssignedTracks();
    updateUnassignedTracks();
    deleteLostTracks();
    createNewTracks();
    displayTrackingResults();

    fprintf('Processed frame %4d. Flag var = %d.\n', nframe, flagv); % Update the user on frame number and flag variable value
end

% Create System Objects
% Create System objects used for reading the video frames, detecting foreground objects, and displaying results.
function obj = setupSystemObjects()
% Initialize Video I/O
% Create objects for reading a video from a file, drawing the tracked objects in each frame, and playing the video.
% Create a video reader.
obj.reader = VideoReader('New Fly_1 p1.mp4');
% Create two video players, one to display the video, and one to display the foreground mask.
%obj.maskPlayer = vision.VideoPlayer('Position', [0, 600, 1920, 1080]);
obj.videoPlayer = vision.VideoPlayer('Position', [600, 0, 800, 600]);
% Create System objects for foreground detection and blob analysis
% The foreground detector is used to segment moving objects from the background.
% the background. It outputs a binary mask, where the pixel value
% of 1 corresponds to the foreground and the value of 0 corresponds
% to the background.

obj.detector = vision.ForegroundDetector('NumGaussians', 3, ...
    'NumTrainingFrames', 20, 'MinimumBackgroundRatio', 0.85);
% Connected groups of foreground pixels are likely to correspond to moving
% objects. The blob analysis System object is used to find such groups
% (called 'blobs' or 'connected components'), and compute their
% characteristics, such as area, centroid, and the bounding box.

obj.blobAnalyser = vision.BlobAnalysis('BoundingBoxOutputPort', true, ...
    'AreaOutputPort', true, 'CentroidOutputPort', true, ...
    'MinimumBlobArea', 1300);

end

%% Initialize Tracks
% The |initializeTracks| function creates an array of tracks, where each
% track is a structure representing a moving object in the video. The
% purpose of the structure is to maintain the state of a tracked object.
% The state consists of information used for detection to track assignment,
% track termination, and display.
% The structure contains the following fields:
%
% * |id| : the integer ID of the track
% * |bbox| : the current bounding box of the object; used
%          for display
% * |kalmanFilter| : a Kalman filter object used for motion-based
%           tracking
% * |age| : the number of frames since the track was first
%        detected
% * |totalVisibleCount| : the total number of frames in which the track
%        was detected (visible)
% * |consecutiveInvisibleCount| : the number of consecutive frames for
%        which the track was not detected (invisible).
%
% Noisy detections tend to result in short-lived tracks. For this reason,
% the example only displays an object after it was tracked for some number
% of frames. This happens when |totalVisibleCount| exceeds a specified
% threshold.
%
% When no detections are associated with a track for several consecutive
% frames, the example assumes that the object has left the field of view
% and deletes the track. This happens when |consecutiveInvisibleCount|
% exceeds a specified threshold. A track may also get deleted as noise if
% it was tracked for a short time, and marked invisible for most of the
% frames.

function tracks = initializeTracks()
% create an empty array of tracks
tracks = struct(...
'id', {}, ...
'bbox', {}, ...
'kalmanFilter', {}, ...
'age', {}, ...
'totalVisibleCount', {}, ...
'consecutiveInvisibleCount', {});
end

%% Detect Objects
% The |detectObjects| function returns the centroids and the bounding boxes
% of the detected objects. It also returns the binary mask, which has the
% same size as the input frame. Pixels with a value of 1 correspond to the
% foreground, and pixels with a value of 0 correspond to the background.
% The function performs motion segmentation using the foreground detector.
% It then performs morphological operations on the resulting binary mask to
% remove noisy pixels and to fill the holes in the remaining blobs.
function [centroids, bboxes, mask] = detectObjects(frame)
% Detect foreground.
mask = obj.detector.step(frame);
% Apply morphological operations to remove noise and fill in holes.
mask = imopen(mask, strel('rectangle', [3,3]));
mask = imclose(mask, strel('rectangle', [15, 15]));
mask = imfill(mask, 'holes');
% Perform blob analysis to find connected components.
[~, centroids, bboxes] = obj.blobAnalyser.step(mask);
end

%% Predict New Locations of Existing Tracks
% Use the Kalman filter to predict the centroid of each track in the
% current frame, and update its bounding box accordingly.
function predictNewLocationsOfTracks()
for i = 1:length(tracks)
    bbox = tracks(i).bbox;
    % Predict the current location of the track.
    predictedCentroid = predict(tracks(i).kalmanFilter);
    % Shift the bounding box so that its center is at
    % the predicted location.
    predictedCentroid = int32(predictedCentroid) - bbox(3:4) / 2;
    tracks(i).bbox = [predictedCentroid, bbox(3:4)];
end
end

%% Assign Detections to Tracks
% Assigning object detections in the current frame to existing tracks is
% done by minimizing cost. The cost is defined as the negative
% log-likelihood of a detection corresponding to a track.
% The algorithm involves two steps:
% Step 1: Compute the cost of assigning every detection to each track using
% the |distance| method of the |vision.KalmanFilter| System object(TM). The
% cost takes into account the Euclidean distance between the predicted
% centroid of the track and the centroid of the detection. It also includes
% the confidence of the prediction, which is maintained by the Kalman
% filter. The results are stored in an MxN matrix, where M is the number of
% tracks, and N is the number of detections.
% Step 2: Solve the assignment problem represented by the cost matrix using
% the |assignDetectionsToTracks| function. The function takes the cost
% matrix and the cost of not assigning any detections to a track.
% The value for the cost of not assigning a detection to a track depends on
% the range of values returned by the |distance| method of the
% |vision.KalmanFilter|. This value must be tuned experimentally. Setting
% it too low increases the likelihood of creating a new track, and may
% result in track fragmentation. Setting it too high may result in a single
% track corresponding to a series of separate moving objects.
% The |assignDetectionsToTracks| function uses the Munkres' version of the
% Hungarian algorithm to compute an assignment which minimizes the total
% cost. It returns an M x 2 matrix containing the corresponding indices of
% assigned tracks and detections in its two columns. It also returns the
% indices of tracks and detections that remained unassigned.
function [assignments, unassignedTracks, unassignedDetections] = ...
    detectionToTrackAssignment()
    nTracks = length(tracks);
    nDetections = size(centroids, 1);
    % Compute the cost of assigning each detection to each track.
    cost = zeros(nTracks, nDetections);
    for i = 1:nTracks
        cost(i, :) = distance(tracks(i).kalmanFilter, centroids);
    end
    % Solve the assignment problem.
    costOfNonAssignment = 30;
    [assignments, unassignedTracks, unassignedDetections] = ...
        assignDetectionsToTracks(cost, costOfNonAssignment);
end
% Update Assigned Tracks
% The |updateAssignedTracks| function updates each assigned track with the
% corresponding detection. It calls the |correct| method of
% |vision.KalmanFilter| to correct the location estimate. Next, it stores
% the new bounding box, and increases the age of the track and the total
% visible count by 1. Finally, the function sets the invisible count to 0.
function updateAssignedTracks()
    numAssignedTracks = size(assignments, 1);
    for i = 1:numAssignedTracks
        trackIdx = assignments(i, 1);
        detectionIdx = assignments(i, 2);
        centroid = centroids(detectionIdx, :);
bbox = bboxes(detectionIdx, :);
% Correct the estimate of the object's location
% using the new detection.
correct(tracks(trackIdx).kalmanFilter, centroid);
% Replace predicted bounding box with detected bounding box.
tracks(trackIdx).bbox = bbox;
% Update track's age.
tracks(trackIdx).age = tracks(trackIdx).age + 1;
% Update visibility.
tracks(trackIdx).totalVisibleCount = ...
    tracks(trackIdx).totalVisibleCount + 1;
tracks(trackIdx).consecutiveInvisibleCount = 0;
end
end

%% Update Unassigned Tracks
% Mark each unassigned track as invisible, and increase its age by 1.
function updateUnassignedTracks()
for i = 1:length(unassignedTracks)
    ind = unassignedTracks(i);
    tracks(ind).age = tracks(ind).age + 1;
    tracks(ind).consecutiveInvisibleCount = ...
        tracks(ind).consecutiveInvisibleCount + 1;
end
end

%% Delete Lost Tracks
% The |deleteLostTracks| function deletes tracks that have been invisible
% for too many consecutive frames. It also deletes recently created tracks
% that have been invisible for too many frames overall.
function deleteLostTracks()
if isempty(tracks)
    return;
end
invisibleForTooLong = 15;
ageThreshold = 8;
% Compute the fraction of the track's age for which it was visible.
ages = [tracks(:).age];
totalVisibleCounts = [tracks(:).totalVisibleCount];
visibility = totalVisibleCounts ./ ages;
% Find the indices of 'lost' tracks.
lostInds = (ages < ageThreshold & visibility < 0.6) | ...
    [tracks(:).consecutiveInvisibleCount] >= invisibleForTooLong;
% Delete lost tracks.
tracks = tracks(~lostInds);
end

%% Create New Tracks
% Create new tracks from unassigned detections. Assume that any unassigned
% detection is a start of a new track. In practice, you can use other cues
% to eliminate noisy detections, such as size, location, or appearance.
function createNewTracks()
centroids = centroids(unassignedDetections, :);
bboxes = bboxes(unassignedDetections, :);
for i = 1:size(centroids, 1)
    centroid = centroids(i,:);
bbox = bboxes(i, :);
    % Create a Kalman filter object.
kalmanFilter = configureKalmanFilter('ConstantVelocity', ... 
    centroid, [200, 50], [100, 25], 100);
    % Create a new track.
newTrack = struct(...
    'id', nextId, ...
    'bbox', bbox, ...
    'kalmanFilter', kalmanFilter, ...
    'age', 1, ...
    'totalVisibleCount', 1, ...
    'consecutiveInvisibleCount', 0);
    % Add it to the array of tracks.
tracks(end + 1) = newTrack;
    % Increment the next id.
nextId = nextId + 1;
end
end

function displayTrackingResults()
% Convert the frame and the mask to uint8 RGB.
frame = im2uint8(frame);
mask = uint8(repmat(mask, [1, 1, 3])) .* 255;
minVisibleCount = 8;
if ~isempty(tracks)
    % Noisy detections tend to result in short-lived tracks.
    % Only display tracks that have been visible for more than
    % a minimum number of frames.
    reliableTrackInds = ...
        [tracks(:).totalVisibleCount] > minVisibleCount;
    reliableTracks = tracks(reliableTrackInds);
    % Display the objects. If an object has not been detected
    % in this frame, display its predicted bounding box.
if ~isempty(reliableTracks)
    % Get bounding boxes.
bboxes = cat(1, reliableTracks.bbox);
    % Get ids.
ids = int32([reliableTracks(:).id]);
    % Create labels for objects indicating the ones for
    % which we display the predicted rather than the actual
    % location.
labels = cellstr(int2str(ids'));
predictedTrackInds = ...
    [reliableTracks(:,).consecutiveInvisibleCount] > 0;
isPredicted = cell(size(labels));
isPredicted(predictedTrackInds) = {' predicted'};
labels = strcat(labels, isPredicted);

% Draw the objects on the frame.
frame = insertObjectAnnotation(frame, 'rectangle', ...
    bboxes, labels);
% Draw the objects on the mask.
mask = insertObjectAnnotation(mask, 'rectangle', ...
    bboxes, labels);
end
end

%% Display the mask and the frame.
% obj.maskPlayer.step(mask);
% obj.videoPlayer.step(frame);
end
end

APPENDIX (B): Budget Breakdown

<table>
<thead>
<tr>
<th>Name</th>
<th>Unit Price</th>
<th>SKU</th>
<th>Total Weight</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sony RX100 IV Camera</td>
<td>$525</td>
<td>-</td>
<td>0.60 lbs</td>
<td>$525</td>
</tr>
<tr>
<td>8' Dowel</td>
<td>$12.98</td>
<td>-</td>
<td>N/A, not a part of the camera frame.</td>
<td>$12.98</td>
</tr>
<tr>
<td>3/4 in. x 4 ft. x 8 ft. Plywood</td>
<td>$52.86</td>
<td>-</td>
<td>Used portion weighed approximately 5.0 lbs</td>
<td>$52.68</td>
</tr>
<tr>
<td>L-Bracket</td>
<td>$2.88</td>
<td>-</td>
<td>Less than 1 lb</td>
<td>$2.88</td>
</tr>
<tr>
<td>Camera extension cord</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>-</td>
<td>-</td>
<td>Approximately 200lbs</td>
<td></td>
</tr>
</tbody>
</table>

Additionally, the L-bracket for camera, wooden plank for camera cart, PVC pipe, and extension cable were all additional costs that were subsequently reimbursed as our budget was still able to cover it. The strikethrough for the vertical supports and camera rails indicates that the materials were necessary for purchasing as J Johnson had enough unistrut for us to use.