PHYSICAL DATA COLLECTION AND TRANSPONDENCE SYSTEM FOR BEETLEWEIGHT COMBAT ROBOTICS ARENA USING COMPUTER VISION

Wolfgang Roettiger
University of Arizona
College of Electrical and Computer Engineering
Tucson, AZ, 85719

Faculty Advisor:
Dr. Michael Marcellin

ABSTRACT

Beetleweight combat robotics tournaments are spectator events in which two robots, weighing less than three pounds, are remotely controlled to mechanically damage and disable one another. The controlled environment of the arena, a square plexiglass box with a wooden floor, enables the monitoring of the fight from all angles using cameras. Using an axonometric viewpoint of the arena from above the fight, image detection software can track the position of each robot throughout the match. Both robots are understood to be roughly three pounds. This means that fundamental physical properties, such as velocity, acceleration, force, and energy in impacts are all approximately measurable. While these are all interesting figures to display on-screen when televising tournaments, this system can also be used to quickly measure the effectiveness of drivetrain and weapon systems in a practical environment. Any metric of a combat robot's performance can be calculated empirically using simple physics, but there are far more factors to consider in an actual fight. This system would enable the approximate measurement of several key metrics of a robot in practice. Moreover, this project provides an opportunity for Wildcat Robotics members at the University of Arizona to gain experience working with a tangible application of computer vision.

INTRODUCTION

The University of Arizona's Wildcat Combat Robotics completed its first year involved in Beetleweight (3 lbs.) in the spring of 2024. The Beetleweight class can be highly competitive due to its relatively low cost and the ready availability of parts. However, the low weight cap and primitive electronics utilized mean that onboard telemetry systems are highly impractical. Thus, it can be challenging to collect key metrics, particularly its velocity. A computer vision system enables the tracking of multiple robots concurrently, given a few known parameters. Kinetic energy is one of the most important metrics in combat robotics, as it is a high indicator of how effective a robot's weapon system is. Knowing that all objects that the computer vision system is tracking will be roughly three pounds, neglecting debris, the kinetic energy of a collision can be easily calculated. This is a measurement that is incredibly impractical to try to calculate empirically due to the inherent chaos of robot combat. In addition to applications in robot design, kinetic energy is an interesting and insightful metric for broadcasters to announce while commentating.

EARLY CONCEPTS

The original conception of the project involved a Raspberry Pi 3 Model B+, a popular, low-cost Linux-based computer. A webcam was to be attached to a 3D-Printed chassis that housed both the camera and the Raspberry Pi. The device would feature a hood to prevent glare against the plexiglass roof of the arena, and would sit near the center, providing a roughly axonometric viewpoint of the arena. The program that would perform all the robot tracking would be written in Python using OpenCV, a popular open-source computer vision library.

The development environment consisted of a Python IDE installed on the Raspberry Pi and a camera attached to the onboard dedicated camera port. This proved quickly to be unwieldy, as all programs ran detrimentally slow on the Raspberry Pi. This significantly slowed progress and proved that the Raspberry Pi platform to be unfit for the program as it stood. Moreover, the unpredictable framerate that the Raspberry Pi captured footage at was inconducive to accurate velocity measurements. At this point, it seemed obvious that higher-fidelity hardware would be necessary for acceptable performance.

After setting up a Python development environment in Visual Studio Code, also using OpenCV, several benefits of creating a program that could be run on a Windows PC became apparent. Primarily, the ability to feed footage into the program via a virtual webcam, such as that in the video streaming software OBS Studio proved remarkably helpful in the development process. Moreover, this means that an event broadcaster can, in real time, switch between any camera connected to the PC that is running the stream. Given the ability to enter user-recorded data for the camera's position, angle, and other key parameters, this maximized the versatility of the program. Better quality, well-lit, and focused camera footage also helps the program track objects more accurately, in contrast to the twenty-dollar webcam that was attached to the Raspberry Pi. On any computer fit to stream a video event, the simple Python script running in the background has a negligible effect on system performance.

SOFTWARE

It is important to preface the software with considerations of what exactly the software should do, and how it should approach the goal of collecting kinetic data pertaining to a moving object on-screen. Pop-science will incline one to pursue any sort of image processing with artificial intelligence, which is not always rightfully regarded as distinct from Computer Vision. Truthfully, Computer Vision is an application of a computer to recognize and interpret image data (Microsoft Azure). Thereby, Computer Vision is a fundamental part of the Artificial Intelligence toolkit, but they are not inextricable. Artificial Intelligence is a specific method of applying computer vision and interpreting data with a mathematical model. With that said, Artificial Intelligence is highly computationally intensive, and not necessary for all applications of computer vision. This project only demands the isolation and tracking of pixels on-screen. There is no demand for actual cognition; the computer does not need to know that it is looking for a robot, or anything about it. It is a safe assumption to assume, within the computer's field of view, that anything that moves must be a combat robot that is roughly three pounds. The computer also already knows where the camera is and can use that to apply concrete mathematics to calculate the velocity and kinetic energy of the mass of "changed pixels" it sees moving across the screen.

The OpenCV library for Python is effectively the sole operating part of this project's software. At the most fundamental level, it allows for frame-to-frame comparison, upon which any tracking method is contingent. It also provides easy ways of superimposing boxes for indicating exactly what the program is "detecting," and what any data it is producing pertains to.

The first concept of the program relied on the separation of color channels within each frame and isolating the robots by their distinct colors. Once this idea is verbalized, it becomes glaringly apparent why this was the wrong approach. First, two robots can easily be the same color. Robots can also be the same color as the arena. Color isolation is also a computationally intensive process, as all manipulations had to be performed on three distinct images, one for each red, blue, and green color channels. If these channels are not separated correctly, which is quite easy to get wrong in an environment with inconsistent lighting, almost no reliable tracking can be done. There are ultimately far too many factors at play when it comes to how colors are really presented in an image.

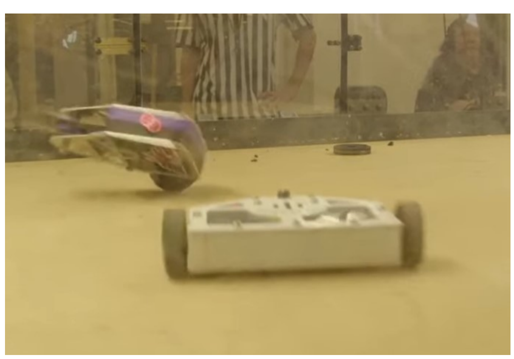

Figure 1 Example of Fight Footage. Note that several objects, namely the floor and the foreground robot, are roughly the same color, and thereby difficult to distinctly track.

With that idea put aside, general motion detection, or image-to-image difference was explored as a method of detecting objects. This detects objects not by trying to isolate any sort of shape, but rather identifying a region of an image that differs from its predecessor. This mitigates any issues pertaining to lighting inconsistencies, or similarities in color between objects that should be regarded as distinct. Using this, several "objects" can be tracked at any given time with reasonable accuracy so long as the camera and the arena do not change more than a threshold of, for example, three-hundred pixels.

With the method chosen, research was conducted to get started in the right direction with code. With no personal experience with Python or any sort of computer vision software, the OpenCV API and a beginner tutorial were consulted to develop the actual program. While it is important to develop a program to a specific project's needs, there are many concepts in computer vision that are universal to all applications thereof, such as feeding footage into the program and accessing differences between frames efficiently.

It is commonplace to convert the image to grayscale in the pursuit of motion detection, as color is completely irrelevant and demands unnecessary computational resources (SokaCoding, 2023). Instead, images can be processed with each pixel containing a single value from 0 to 255. Then, an algorithm is run to simply compare the current frame to the previous frame and find any differences that exceed a pixel-count threshold.

Another important consideration is the angle at which the camera is recording the arena. Assuming the camera is roughly axonometric, meaning the field of view has a sufficiently insignificant effect on the image it collects, the calculation is as shown below in Figure 2.

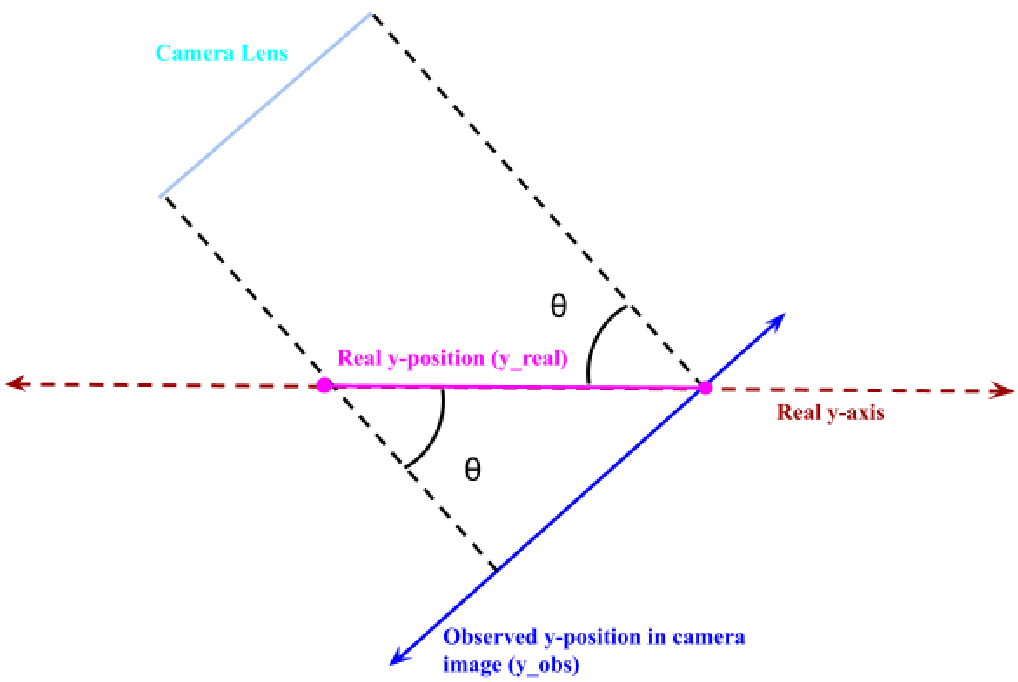

Figure 2 Calculation to approximate the real y-position given a known camera angle.

As shown, the real y position, y_{real} can be found by simple algebraic manipulation:

$$y_{real} = \frac{y_{obs}}{\sin\theta}$$

Thus, any position observed relative to the field of view of the camera, y_{obs}, can be mapped to a value for y_{real} in the real plane. Since the x-axis remains unchanged, and based on the assumption that the camera's viewpoint is axonometric, no manipulations need be done to the x-axis, as the real and observed image planes intersect on the x-axis. This can be visualized by projecting Figure 2 in three dimensions. While any values produced by this method are subject to lens distortion, which occurs since camera lenses are round, but all must be considered within the scope of this project. When the tracking is only accurate to a few pixels, which may correspond to significant fractions of feet, one must understand that this is by no means a precise measurement of kinetic energy, nor is any measurement in this project unmarred by the inherent unpredictability of the environment at hand.

To scale pixels to meters, it is understood that the standard arena for combat robotics at this scale is 8 feet long. Therefore, one can use an image manipulation program to measure exactly how many pixels are occupied on-screen by the arena. This could certainly be expanded upon in a later release by recognizing the edges of the arena by color and scaling the measurements observed in pixels accordingly.

RESULTS

The resultant program is, overall, relatively successful and arguably useful. While it does have some issues with continuity when it loses tracking for a moment, it is relatively easy to isolate the largest recorded value for the kinetic energy of a given robot. Its fatal flaw is that it does not have a perfect record of recognizing a lost object when it is found moving onscreen again.

What is meant by this is best understood in the context of basic computer vision, as discussed before. The computer does not inherently know what it is looking at. It uses a function, findContours(). This looks at a frame which describes the difference between the two images in question: the previous frame and the current frame. It does this using an image composed of what can be thought of as edges of the moved pixels. This is called the threshold frame because this observed difference in pixels is calculated by superimposing the previous frame on the current frame and computing a black and white threshold, showing white where there are differences in the images (Intel). The findContours() function then populates an array with the positions of all of the white artifacts in the threshold image, and draws a box over them in the real image output. While this does track everything on screen, there is no present concept of object permanence. The objects, as perceived, do not necessarily exist outside of the exact frame in which they occur. One can hope that an object will appear in roughly the same area of the screen, such that it populates the same index in the array based simply on how the findContours() function scans through the image, but this is a flawed assumption.

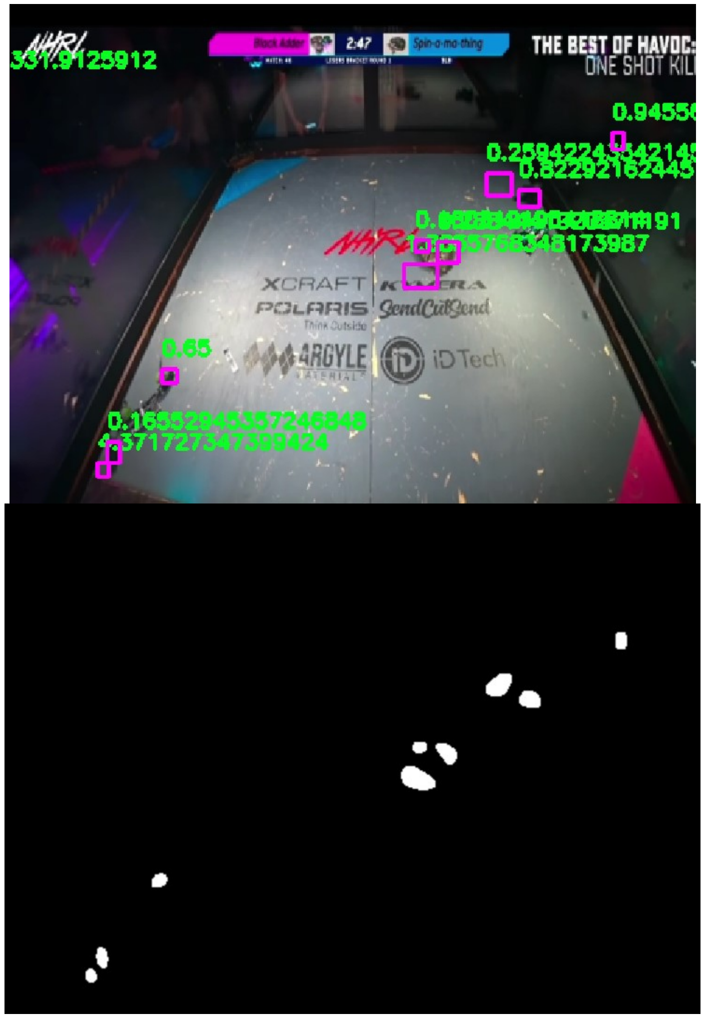

Figure 3 Example of Fight Footage from NHRL Tournament and its Threshold Image (NHRL).

For this reason, the program is only effective when there is one object on the screen. The program also handles noise and texture in images poorly, but there are methods of mitigating this. One can change the amount by which the contours found in the threshold image are dilated, such that holes are filled in and spotty artifacts in the same area can be regarded as one contiguous object.

Ultimately, as shown in the top corner of Figure 3, the maximum recorded kinetic energy for a collision was about 330 Joules, which is roughly consistent with what one would expect. The absolute maximum energy that most weapons store in Beetleweight combat robotics is typically around 800 Joules, so for a bit less than half of that to be imparted to an opponent in a collision is expectable. This is because both robots, at the same mass, should take on about half of the energy of the collision.

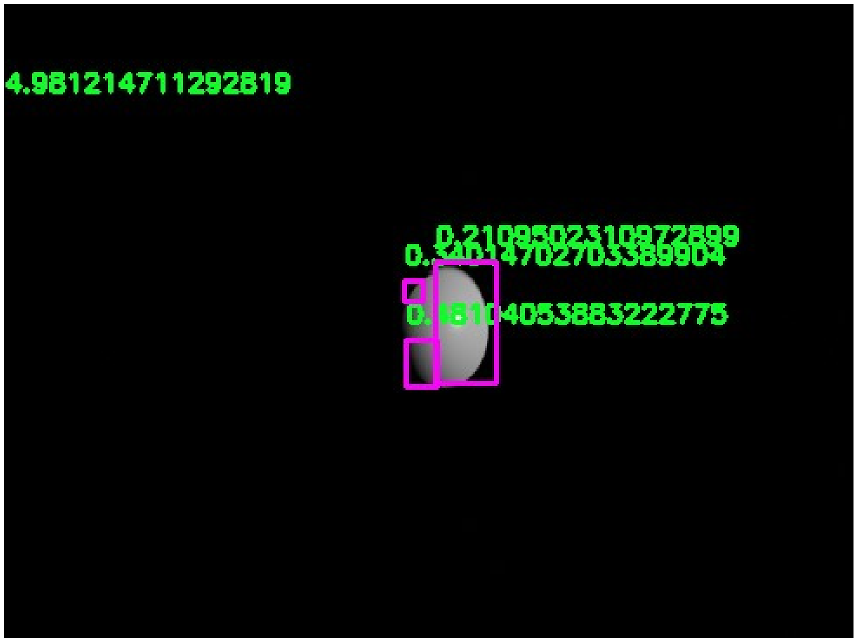

Figure 3 Test footage rendered in Blender to test isolation of simple, uniform shapes.

Given the circumstances and its application, this project does what it is intended to within some reasonable confines. While one could certainly desire far more precise tracking, this project exhibits a fundamental limitation of using pure computer vision to track objects in real time.

IMPLICATIONS

Naturally, one would immediately assume that AI would be far better suited to this application, but that's a far muddier idea than it lets on. AI uses elaborate models to detect patterns in images, which are representations of real objects. It takes a lot of data to get a neural network to recognize an object because it must see it at a multitude of angles and in many different contexts. In combat robotics, contestants are typically discouraged from building robots that are identical to each other, or even similar in structure. This would be quite difficult for a neural network to make sense of, because all it sees is a two-dimensional array of colors. From that viewpoint, virtually every combat robot would demand an entirely different, or incredibly advanced model to recognize with any accuracy. Furthermore, this sort of modeling demands impressive computational power to do in real time, whereas the original intent of this project was to make it accessible and lightweight.

FURTHER RESEARCH

One rather important system that was not explored in-depth is one which distorts the camera image such that the perspective of the camera does not apply inaccurate scaling to the image due to lens distortion. The method of accounting for the incident angle of the camera relies on the assumption that the camera's lens distortion is negligible, which was a fair assessment for the footage used, but not for all cameras. This system was not implemented because the field of view of a camera is not always available, nor is it often entirely influential to an application like this.

Cameras with wide angle, or "fisheye" lenses suffer particularly from "barrel distortion" (Fermum). These lenses are often used to capture areas that are larger than what a human eye could see from the same perspective. These lenses are often used in combat robotics, such that the entire arena can be captured.

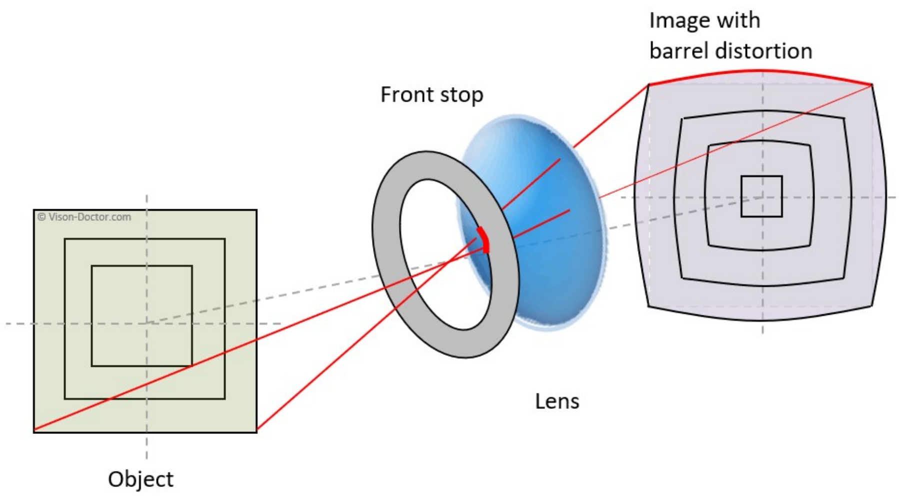

Figure 4 Graphical representation of barrel distortion (Fermum).

There are several ways to approach this. The simplest of which is to simply calculate a distortion transformation to apply to the image before it is processed. This can be calculated given some key metrics about the camera in use, such as the lens field of view. Another more advanced way to do this makes use of a unique property of combat robotics. The arena itself is square, therefore the edges of the arena should always appear straight. One could write a computer vision function that runs when the program starts that looks for an edge in the image. This could be done similarly to how the moving objects were recognized. The change in darkness at the edge of a solid can be used to detect its presence. Based on the perceived curve of this object, which could be modeled by a logarithm, the corrective transformation could be generated and applied.

Unfortunately, this process alone might not be able to meaningfully affect the precision of measurements, because this transformation inherently changes the shape, and therefore the area occupied by a robot on-screen. Just as the fisheye lens "stretches out the center, and makes it appear larger, the actual image seen by the camera does not become orthogonal. It still sees everything from a wide angle, which means that objects will not be seen as physically rotated, like they ideally would be. Instead, their profiles from the camera's viewpoint are distorted, potentially making objects appear larger or smaller than they truly are. Since the position of objects detected in the program is determined by the midpoint of the cluster of pixels it sees, this would change the perceived position of the object, altering its perceived velocity.

CONCLUSIONS

This project implemented a computer vision system to estimate metrics in real-time in a combat robotics arena. The system successfully recognized images to a reasonable degree of reliability and was able to track their motion across the camera's field of view. This has multiple applications, the foremost being for testing purposes. Since it is difficult to empirically calculate the real kinetic energy imparted by a weapon's contact in combat robotics, this is a practical way to see how much energy is involved in any collision. This can also be used by commentators to provide facts to viewers and provide a more engaging experience. While the tracking system is not perfect, this exemplifies a real limitation of computer vision. When given an environment that has a lot of noise, or data that must be filtered out, it can be uniquely challenging to collect reliably accurate and precise data. In the future, artificial intelligence concepts could be experimented with to more reliably perform these operations, but there are drawbacks to both pure computer vision algorithms and neural network-based approaches.

ACKNOWLEDGEMENTS

We would like to thank Dr. Michael Marcellin for his sustained commitment to providing opportunities for University students to undertake real engineering challenges and his indispensable support for the University of Arizona's Wildcat Combat Robotics team.

CITATIONS AND OTHER REFERENCES

Fermum, L. (n.d.). *Optic distortion*. Distortion. https://www.vision-doctor.com/en/optical-errors/distortion.html

Intel. (n.d.). *OpenCV modules*. OpenCV. https://docs.opencv.org/4.x/

Microsoft Azure. (n.d.). *What is Computer Vision?: Microsoft Azure*. What Is Computer Vision? | Microsoft Azure. https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-computer-vision#object-classification

NHRL. (2022, February 18). *One shot kill. total and instant, robot destruction.* YouTube. https://www.youtube.com/watch?v=bxT9f5LwXaA

Sokacoding. (2023, January 24). *Simple motion detection with python and OpenCV-for Beginners*. Medium. https://sokacoding.medium.com/simple-motion-detection-with-python-and-opencv-for-beginners-cdd4579b2319