Utilizing computer vision to inform real-time simulations and controlling embedded systems using various wireless techniques.

Author: Chauncey Deone Hester Jr.
University of Kansas
Lawrence Kansas, 66045

Faculty Advisor: Dr Erik Perrins

Abstract

The field of robotics is filled with many interwoven concepts, each with their own niche yet relevant function which serves a higher abstracted purpose. At the University of Kansas Robotics Organization (KURO), we offer students the freedom to explore these underlying concepts in self contained and interesting projects. As such, in this paper, we will recount the methodology and results of combining a wide array of topics together for an insightful yet novel goal of recreating a carnival game with a twist: A shooting gallery but your finger is the gun. Such topics include web socket communication, advanced hand tracking hardware, game development software, RF communication between multiple embedded systems, and real time computer vision. Along with this, but not necessarily relevant to Telemetry, we will discuss the manufacturing considerations, for optimizing speed and stability, as the entire project had a strict time constraint of 48 hours to create a working demo to be judged.

Key Words

Real-Time Simulation, Short Range RF Communication, Web Socket Communication, Embedded Systems, Computer Vision, Case Study.

Introduction

This paper will discuss the KU Robotics Organizations attempt to design and implement a novel carnival game, a Shooting Gallery, within a 48-hour timeframe for a judged software competition. Our methodology, struggles, and results of this time constrained project will be covered, as well as where the project has gone following this exercise.

The project covers a wide array of topics, most of which were new to us. Such topics are represented as the following tasks: use computer vision to extract positional information of unique targets from a picture, use this information to craft a virtual representation in a physics engine, enable user input in this simulation via commercial hand tracking hardware, maintain persistent real time communication between the simulation and the deployed embedded system with web sockets, and seamlessly have state changes in the simulation affect one of the deployed target via RF communication. Along with these tasks, all the manufacturing of the prototypes had to be done in 48 hours as well, and as these real-world constraints directly affected the desired product, this paper will cover the methodology and techniques we used to rapidly prototype before the deadline of the demo.

This paper will cover each aspect of the project in depth, starting with relevant background information, before moving onto the top-level project overview. Each topic covered in the overview will be expanded upon in its own relevant section in this paper. After this, the judging and conclusion will explore where we plan to take this project in the future, as with our organization, no project is ever truly finished, only further refined and maintained for future members to learn form.

Background

With Robotics being a generic term for a vast field, The University of Kansas has no wholistic program for learning about these topics in depth. As of now, pieces of this larger puzzle can be gleaned through degrees such as Computer Science, Computer Engineering, Electrical Engineering, Mechanical Engineering, Aerospace Engineering, and Engineering Physics. Furthermore, within these degrees, the opportunities for students to gain firsthand experience remains limited to a few select courses and labs within those programs.

These conditions led to the founding of The University of Kansas Robotics Organization (KURO) in February 2018. The organization serves as a linking point between these normally separate degrees, as well as an opportunity for motivated students to immediately gain hands on experience in whatever topics interest them within the budgetary constraints of the club. In just 4 years, the club has amassed a large wealth of knowledge and resources, all of which, in turn, feed back into itself, allowing the organization to tackle ever greater projects while still nurturing new members into self-sufficient, uniquely well-rounded, and multidisciplinary Engineers, regardless of their baseline knowledge at the time of joining the organization.

The organization, since its founding, has maintained an annual tradition of jumpstarting large scale projects by taking part in the KU's yearly Hackathon, HackKU, hosted by the KU chapter of The Association for Computing Machinery. The event is sponsored by Major League Hacking, as well as other companies. While the addition of manufacturing and hardware in a traditionally software-oriented event has led to poorer performance in the judging component of the competition, it also forces us to tackle normally daunting projects few other undergrads would ever consider, leading to us producing some impressive flagship projects in the following semesters of refinement from the initial concept prototyped in 48 hours.

This year's project was born from having a collection of concepts we wanted to consider learning about and the competition having no theme to rule out any. So, after about 2 hours of deliberation on what we would even make, the Shooting Gallery project was decided upon. The group working on this project included Jordan Hirsekorn, a recent Electrical Engineering Alumni from our organization and former executive member; Ian Kim, an experienced, current executive member of KURO, and Aerospace Engineer; Caleb Zinabu, a recent member to join KURO and Computer Engineering Sophomore; and Chauncey Hester, a 5th Year Computer Engineering Undergrad, and founding President of the Organization. The event hosted by the professional organization for Computer Scientists at KU, and not one person in our group was even a Computer Scientist. A typical occurrence in KURO.

Project Overview

This section of the paper covers all the top-level goals of the Shooting Gallery. In a traditional shooting gallery, multiple targets will spring up, and the player will use a toy gun to shoot pellets at them, knocking the targets down in the process. Our project sought to recreate this game without using any prop toy and merely the user's hand in a finger gun gesture. Given the pressing time constraint, we identified the minimum condition of a working game to be being able to locate and identify a minimum of 5 distinct targets, creating a rough enough virtual representation that feels relatively correct, and pointing at targets causing at least a one state change within a reasonable time for judging. In figure 1, we show the derived interaction diagram used to plan out what the overall project's system would look like.

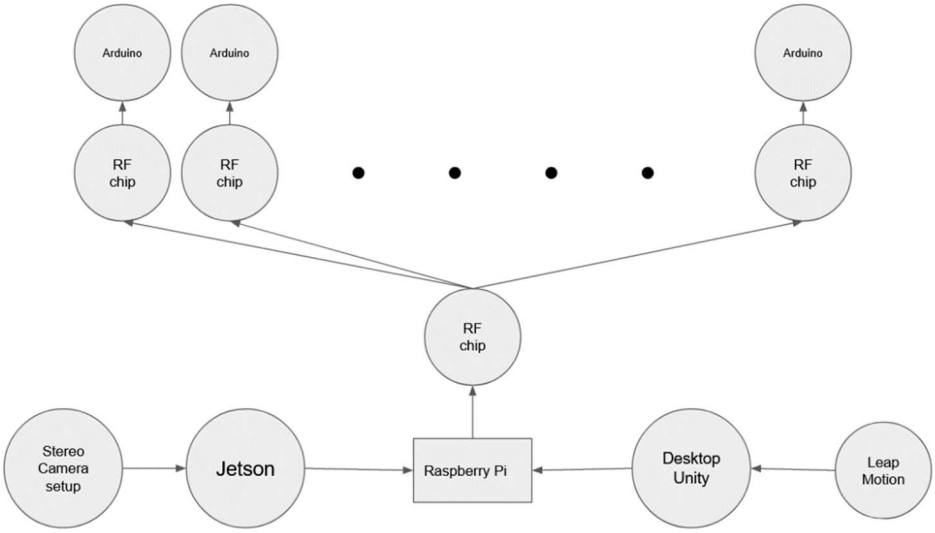

Figure 1: Interaction Diagram

As shown, we made the decision to use the NVIDIA Jetson Nano development board [1] to perform computer vision separate from the primary embedded system. This decision lay purely in the division of labor for development, as the Jetson Nano is more than capable for acting as the primary system's microcomputer.

This interaction diagram, created in the first few hours of the project's inception, would also be used in fully planning out the tasks needed to be performed in each system, as will be further described below in each relevant section.

Creating the Unity Game

This section goes more into depth with the virtual simulation, created using the Unity Game Engine [2]. This engine was used as a solution to a pressing question: How does one quickly and accurately get the orientation of a user's index finger and determine if its pointing at something. Many potential solutions were thrown around, and we ultimately wanted to experiment with using a consumer product known as a Leap Motion Controller [3], which has plug ins compatible

with Unity, as well as some form of usability with Arduino. This part of the overarching system functioned as such: Upon opening the game, present an interface for inputting the primary embedded system's IP address and attempt to connect. Upon successful connection, enter a setup state and await a data packet containing all parameters of the virtual space. Such parameters include the Leap Motion table height, number of targets, each target's Unique ID, each target's XY coordinate and estimated depth. With each target having fixed dimensions, these parameters would be enough to enable a user to point at one and the unity game report which box was being pointed at.

Using Unity was both a blessing and a curse, in that it functioned as an extremely well documented engine, however, with it also came all the struggles of trying to develop a game and debug it in real time using nothing but a simple log tool. The Leap Motion controller also performed all the calculations on its board, allowing us to skip a potentially costly venture in deriving that math ourselves. However, the controller has an extremely limited visible range and can only get the pose information when not obscured from below. Meaning hands cannot overlap, and the fingers can potentially be incorrect in certain arm positions. Figure 2 shows the view from the Leap's point of view and the estimated fingers. Using the built-in library for the Leap Motion Controller, getting the vector of the Index Fingertip was a straightforward process. Figure 3 shows the game view prior to exporting the game as an executable file.

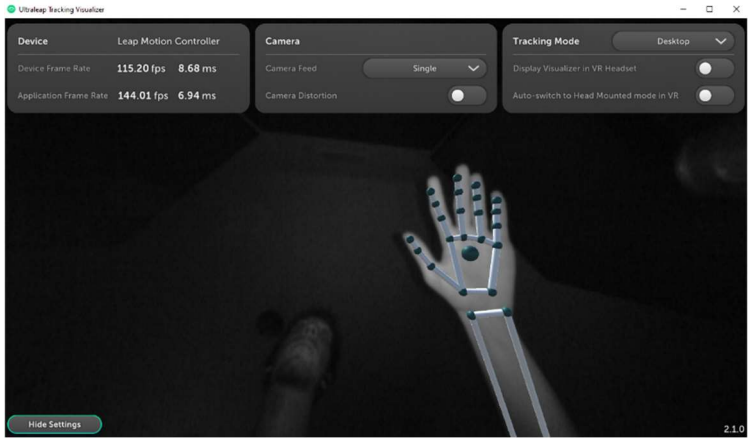

Figure 2: Leap Motion Controller View

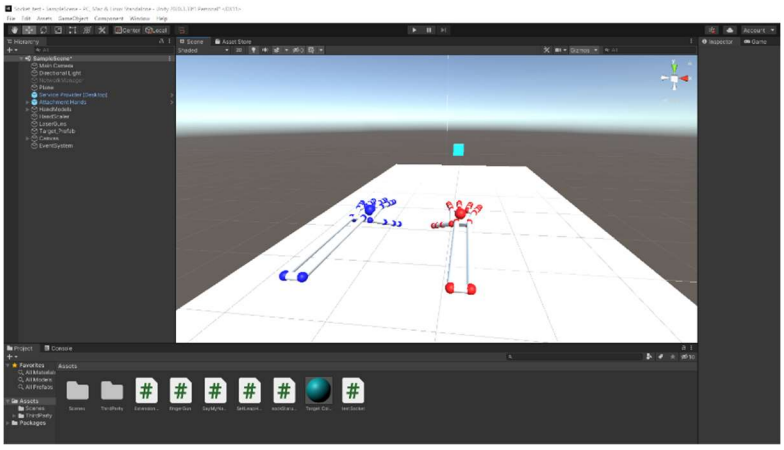

Figure 3: Unity Game View

The most pressing concern with using Unity this way was the lack of dimensions. All the assets had to be measured rigorously and scaled relative to the Leap Motion Controller's virtual hand asset. Which presented a unique challenge of needing to flexibly alter assets as designs and environment constraints changed, as they would not be finalized until much later in the 48-hour period.

Web Socket Interface between Unity & Raspberry PI

This section explains the Web socket information between the primary embedded system for shooting gallery. This topic was born from the growing need to remote operate robots in the organization using methods other than direct connection with the onboard Bluetooth module on the Raspberry Pi 4 [4]. Past attempts to use SSH to stream a GUI using X forwarding, while somewhat successful, immediately presented a bottleneck in data transfer speeds, as well as lacking user friendliness in non-Linux based machines. Web sockets were considered as an alternative, however that required a second script to be running in parallel with its own accompanying GUI, which would potentially be disproportionally time consuming to set up relative to the scope of the project.

Unity was offered as a potential fix to all these problems, as well as conveniently satisfying the need to perform a user driven simulation with the Leap Motion controller. After going through the documentation on how to set up Sockets both in Unity's C# and on the Raspberry Pi's Python Socket library, one way communication was easily established, although 2-way communication posed a more substantial challenge, as well as error handling on the Unity side. The data being transferred was split into two parts. The initial setup data packet from the primary embedded system to the Desktop, and the state change information when the user points at a target or moves their hand away from that target.

Extracting Real World data in Real-time Computer Vision

This section of the paper discusses the efforts to use OpenCV [5] to extract positional and categorical information for the targets within the frame**.** This was another new venture for KURO, as interacting with computer vision projects thus far had only involved using existing pre-trained image detection models for simple things like face recognition, pet detection, etc. The goal was simple, use a PiCamera [6] to take a photo of the play area and use computer vision to find the targets within the frame, and identify their ID using the front panel which would have a separate color based on an RGB LED behind the semitransparent material.

To accomplish this, we decided to design our box to have an easily recognizable color, then tweak the OpenCV code to reliably find that color using the built-in contour detection function and checking the largest shapes to see if they were this known color within some margin of error. While not a robust solution, it certainly felt far more feasible for complete beginners to pull off in 48 hours. Figure 4, 5, and 6 shows the results of this methodology.

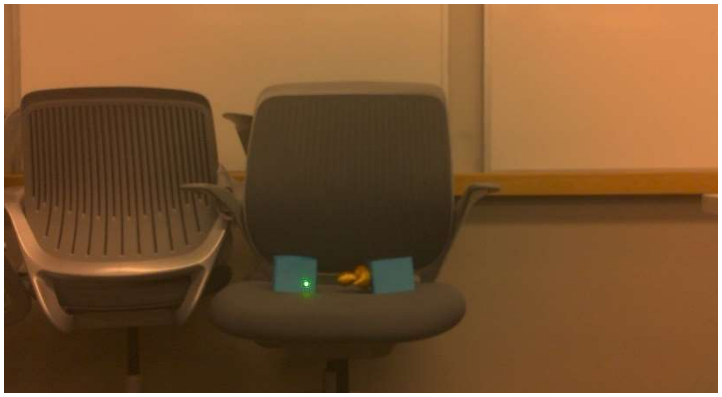

Figure 4: Initial Camera Picture

Figure 5: Detected Contours

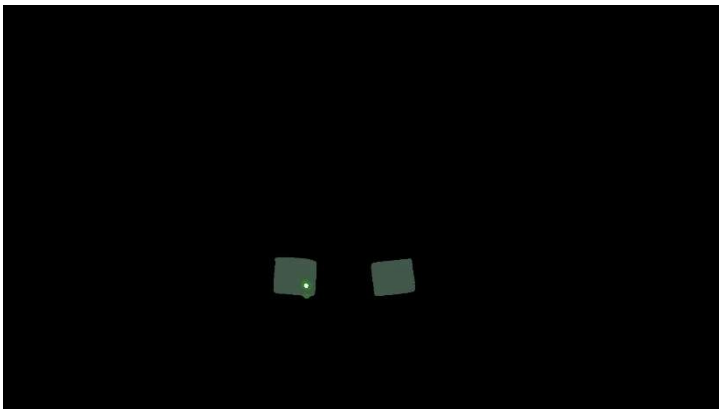

Figure 6: Isolated Target

One crucial, but unfortunately unexplored element to our computer vision system was the inability to reliably to get any form of depth information from a singular picture, which would be required for the virtual system to properly place its targets. Stereographic camera setups were considered to use OpenCV's Disparity Mapping function to estimate depth, but time constraints ultimately forced us to abandon exploring that avenue during the 48 hours. Instead, an estimate was gathered from the relative size of the detected targets that where offset from a single target with a known distance from the camera and table. Given our limited mathematical understanding of ocular distortion, this was our best guess at how to solve the depth issue.

RF communication between Raspberry PI and Deployed Embedded Systems

This section of the paper covers the interaction between our primary embedded system and the target sub systems. The target systems were designed to be a simple deployable structure and facade, with internals consisting of an Arduino Nano microprocessor [7], a multi-channel RF chip [8], a driven servo motor platform, a simple yellow duck target to mount on said platform, a single RGB LED for visual identification, and a 9v power supply. Figure 7 shows these internals wired up to the Arduino Nano.

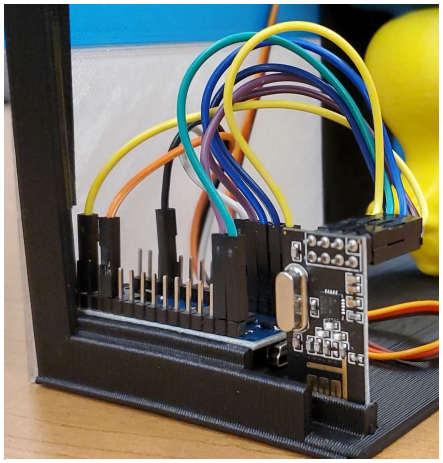

Figure 7: Arduino Wired Up with RF Chip

Upon receiving power, each system has a predefined unique ID and instructions to enter a setup state with its LED on and servo set to 90 degrees. Given its unique ID, it will set up its individual RF chip to wait on its desired channel for further instructions after verifying 2-way communication with the primary system. After the primary system is done identifying each target, the command is sent to each target to turn off their LED and await commands to drive their servo platform. Given the straightforward nature of this section, it was saved for the later portion of the 48 hours and thus the only issues encountered consisted of trying to wire the RF chip correctly to the Arduino while sleep deprived. Luckily, nothing was damaged in the process.

Manufacturing Considerations and Techniques

This section covers the manufacturing aspect of the project. To accomplish this project's goals in 48 hours, speed and stability had to be balanced. As failed 3D prints was equally if not more devastating compared to printing slower. In total, we had four 3D printers running concurrently, although each printer had varying levels of print quality, so complicated and more error prone prints were done on the higher end printer. The Artillery Sidewinder X1 [9] was our fastest and best tuned printer. It prioritized the structural supports of the system and printing the front facades. The Flashforge Creator Pro [10] was dependable, albeit aged, and was tasked with printing the rotating servo platforms and some internal structures. The Lulzbot Sidekick [11] is our newest printer that arrived the same day as the competition, thus it was untuned, so we only printed the duck targets and a few face panels. The Robo R2 [12] was a bit unreliable due to its damaged build plate, thus it only worked on making ducks.

Various optimizations were made to aid in optimizing print speed and stability. Our CAD designs were made with as few overhangs as possible, and existing overhangs needing as little supports as possible during the print. Figure 8 shows our initial CAD design, made in Siemens NX [13], using these design principals.

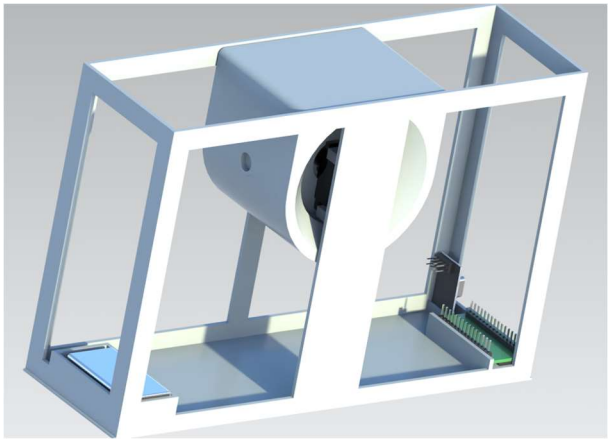

Figure 8: CAD System

Along with this, effort was put into our slicer parameters, using CURA Slicer Ver 1.17 [14], to improve speed as much as possible without compromising on integrity. Line width was always set to larger than our nozzle width. We printed at higher temperatures than manufacturer labeled to enable us to raise the flow rate settings on the printer. The servo platforms were printed using variable infill to reduce on print time. The ducks were printed using tree style supports, which are faster and more efficient, albeit best suited for printing organic surfaces. Lastly the tops and bottoms of the of non-visual structural prints were removed to save time as well.

Despite our best efforts, however, problems were always bound to happen when trying to high-speed print. The extruder motor has a finite strength, which limits how much extruder throughput is possible. The stepper motors would also occasionally move faster than the extruder could output, leading to some skipping. And lastly, some components have strict tolerances for mounting components on, and thus cannot be accelerated beyond a certain speed whilst maintaining that constraint. Within 48 hours, we were able to complete 6 targets, one pictured below in figure 9.

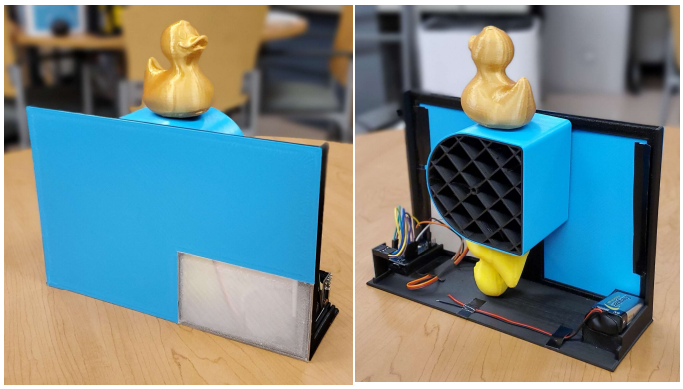

Figure 9: Realized System Front and Back

Judging, Further Considerations, and Conclusion

After 48 hours, our organization learned a great deal about the topics we initially set out to explore, and far more progress was made than expected. All the interwoven parts functioned to some degree in a vacuum. However, there were unforeseen challenges in the runtime flow when trying to put the systems together into a unified project. The project demo was instead given showing how each topic we wished to explore worked in isolation. So, while we won no accolades for our efforts, we walked away from the event far more experienced than 48 hours prior.

Further considerations for this project include: Adding better error handling to account for runtime errors due to the many points of failures in the overall system, removing the redundant Raspberry Pi 4 as the primary system microcomputer and using the Jetson Nano for both that, and the Computer Vision system, Adding a comprehensive UI to the Unity game to improve user experience, and expanding our computer vision setup to use a stereographic setup to attempt to extract depth information from the play area.

Given the circumstances, this project can be deemed an immense success for the organization, as it keeps in line with our founding principles of gaining as much hands-on experience as possible, while also existing as an interesting concept to get many more aspiring Engineers excited for what's possible in merely 48 hours of nonstop effort.

References

1. NVIDIA Jetson Nano Documentation: https://docs.nvidia.com/jetson/

2. Unity Game Engine Documentation: https://docs.unity3d.com/Manual/index.html

3. Leap Motion Controller: https://www.ultraleap.com/product/leap-motion-controller/

4. Raspberry Pi 4: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

5. OpenCV: https://docs.opencv.org/4.x/

6. PiCamera: https://www.raspberrypi.com/products/camera-module-v2/

7. Arduino Nano: https://docs.arduino.cc/hardware/nano

8. RF Chip: https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf

9. Artillery Sidewinder X1: https://artillery3d.com/products/artillery-sidewinder-x1-sw-x1-3d-printer-300x300x400mm-large-plus-size-high-precision-dual-z-axis-tft-touch-screen

10. Flashforge Creator Pro: https://www.flashforge.com/product-detail/flashforge-creator-pro-3d-printer

11. Lulzbot Sidekick: https://lulzbot.com/store/sidekick289?ref=null

12. Robo R2: https://robo3d.com/blogs/news/anatomy-of-the-robo-r2-3d-printer
13. Siemens NX: https://www.plm.automation.siemens.com/global/en/products/nx/

14. Cura Slicer: https://ultimaker.com/software/ultimaker-cura